Developer Guide

Amazon Elastic Container Service

Amazon Elastic Container Service: Developer Guide

Table of Contents

What is Amazon Elastic Container Service?

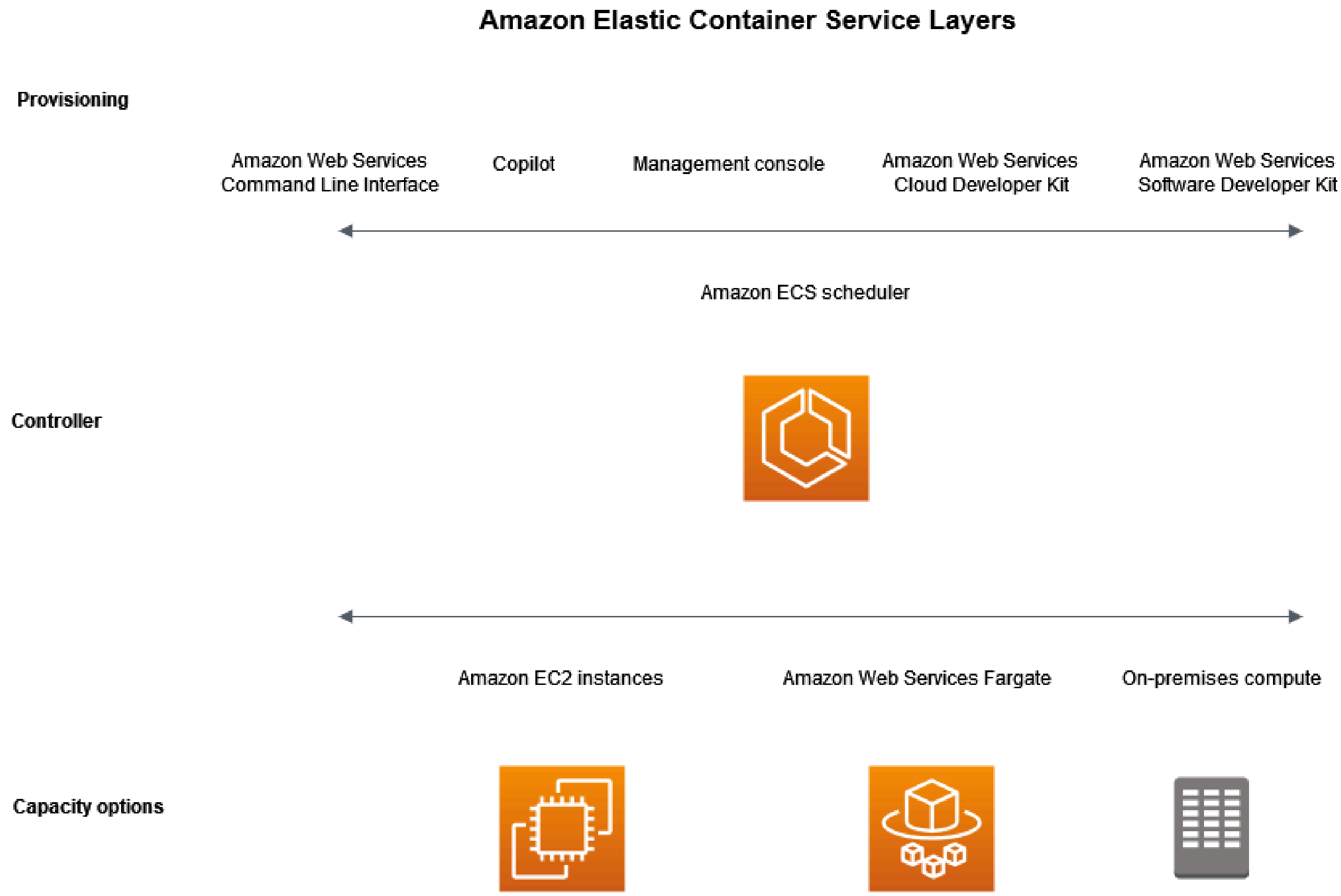
Amazon Elastic Container Service (Amazon ECS) is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized applications. As a fully managed service, Amazon ECS comes with AWS configuration and operational best practices built-in. It's integrated with both AWS tools, such as Amazon Elastic Container Registry, and third-party tools, such as Docker. This integration makes it easier for teams to focus on building the applications, not the environment. You can run and scale your container workloads across AWS Regions in the cloud, and on-premises, without the complexity of managing a control plane.

Terminology and components

There are three layers in Amazon ECS:

- Capacity - The infrastructure where your containers run
- Controller - Deploy and manage your applications that run on the containers
- Provisioning - The tools that you can use to interface with the scheduler to deploy and manage your applications and containers

The following diagram shows the Amazon ECS layers.



The capacity is the infrastructure where your containers run. The following is an overview of the capacity options:

- Amazon EC2 instances in the AWS cloud

  You choose the instance type, the number of instances, and manage the capacity.
- Serverless (AWS Fargate) in the AWS cloud

  Fargate is a serverless, pay-as-you-go compute engine. With Fargate you don't need to manage servers, handle capacity planning, or isolate container workloads for security.
- On-premises virtual machines (VM) or servers

  Amazon ECS Anywhere provides support for registering an external instance such as an on-premises server or virtual machine (VM), to your Amazon ECS cluster.

The Amazon ECS scheduler is the software that manages your applications.

Features

Amazon ECS provides the following high-level features:

Task definition

The blueprint for the application.

Cluster

The infrastructure your application runs on.

Task

An application such as a batch job that performs work, and then stops.

Service

A long running stateless application.

Account Setting

Allows access to features.

Cluster Auto Scaling

Amazon ECS manages the scaling of Amazon EC2 instances that are registered to your cluster.

Service Auto Scaling

Amazon ECS increases or decreases the desired number of tasks in your service automatically.

Provisioning

There are multiple options for provisioning Amazon ECS:

- **AWS Management Console** — Provides a web interface that you can use to access your Amazon ECS resources.
- **AWS Command Line Interface (AWS CLI)** — Provides commands for a broad set of AWS services, including Amazon ECS. It's supported on Windows, Mac, and Linux. For more information, see AWS Command Line Interface.
- **AWS SDKs** — Provides language-specific APIs and takes care of many of the connection details. These include calculating signatures, handling request retries, and error handling. For more information, see AWS SDKs.

- **AWS CDK** — Provides an open-source software development framework that you can use to model and provision your cloud application resources using familiar programming languages. The AWS CDK provisions your resources in a safe, repeatable manner through AWS CloudFormation.

Pricing

Amazon ECS pricing depends on the capacity option you choose for your containers.

- Amazon ECS pricing – Pricing information for Amazon ECS.
- AWS Fargate pricing – Pricing information for Fargate.

Related services

Services to use with Amazon ECS

You can use other AWS services to help you deploy yours tasks and services on Amazon ECS.

Amazon EC2 Auto Scaling

Helps ensure you have the correct number of Amazon EC2 instances available to handle the load for your application.

Amazon CloudWatch

Monitor your services and tasks.

Amazon Elastic Container Registry

Store and manage container images.

Elastic Load Balancing

Automatically distribute incoming service traffic.

Amazon GuardDuty

Detect potentially unauthorized or malicious use of your container instances and workloads.

Learn how to create and use Amazon ECS resources

The following guides provide an introduction to the tools available to access Amazon ECS and introductory procedures to run containers. Docker basics takes you through the basic steps to create a Docker container image and upload it to an Amazon ECR private repository. The getting started guides walk you through using the AWS Copilot command line interface and the AWS Management Console to complete the common tasks to run your containers on Amazon ECS and AWS Fargate.

Contents

Set up to use Amazon ECS

If you've already signed up for Amazon Web Services (AWS) and have been using Amazon Elastic Compute Cloud (Amazon EC2), you are close to being able to use Amazon ECS. The set-up process for the two services is similar. The following guide prepares you for launching your first Amazon ECS cluster.

Complete the following tasks to get set up for Amazon ECS.

AWS Management Console

The AWS Management Console is a browser-based interface for managing Amazon ECS resources. The console provides a visual overview of the service, making it easy to explore Amazon ECS features and functions without needing to use additional tools. Many related tutorials and walkthroughs are available that can guide you through use of the console.

For a tutorial that guides you through the console, see Learn how to create and use Amazon ECS resources.

When starting out, many customers prefer using the console because it provides instant visual feedback on whether the actions they take succeed. AWS customers that are familiar with the AWS Management Console, can easily manage related resources such as load balancers and Amazon EC2 instances.

Start with the AWS Management Console.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Create a virtual private cloud

You can use Amazon Virtual Private Cloud (Amazon VPC) to launch AWS resources into a virtual network that you've defined. We strongly suggest that you launch your container instances in a VPC.

If you have a default VPC, you can skip this section and move to the next task, Create a security group. To determine whether you have a default VPC, see Work with your default VPC and default subnets in the *Amazon VPC User Guide*. Otherwise, you can create a nondefault VPC in your account using the steps below.

For information about how to create a VPC, see Create a VPC in the *Amazon VPC User Guide*, and use the following table to determine what options to select.

Option	Value
Resources to create	VPC only
Name	Optionally provide a name for your VPC.
IPv4 CIDR block	IPv4 CIDR manual input The CIDR block size must have a size between /16 and /28.
IPv6 CIDR block	No IPv6 CIDR block
Tenancy	Default

For more information about Amazon VPC, see What is Amazon VPC? in the *Amazon VPC User Guide*.

Create a security group

Security groups act as a firewall for associated container instances, controlling both inbound and outbound traffic at the container instance level. You can add rules to a security group that enable you to connect to your container instance from your IP address using SSH. You can also add rules that allow inbound and outbound HTTP and HTTPS access from anywhere. Add any rules to open ports that are required by your tasks. Container instances require external network access to communicate with the Amazon ECS service endpoint.

If you plan to launch container instances in multiple Regions, you need to create a security group in each Region. For more information, see Regions and Availability Zones in the *Amazon EC2 User Guide*.

> **Tip**
>
> You need the public IP address of your local computer, which you can get using a service. For example, we provide the following service: http://checkip.amazonaws.com/ or https://checkip.amazonaws.com/. To locate another service that provides your IP address, use the search phrase "what is my IP address." If you are connecting through an internet service provider (ISP) or from behind a firewall without a static IP address, you must find out the range of IP addresses used by client computers.

For information about how to create a security group, see Create a security group for your Amazon EC2 instance in the *Amazon EC2 User Guide* and use the following table to determine what options to select.

Option	Value
Region	The same Region in which you created your key pair.
Name	A name that is easy for you to remember, such as *ecs-instances-default-cluster*.
VPC	The default VPC (marked with "(default)"). **Note** If your account supports Amazon EC2 Classic, select the VPC

Option	Value
	that you created in the previous task.

For information about the outbound rules to add for your use cases, see Security group rules for different use cases in the *Amazon EC2 User Guide*.

Amazon ECS container instances do not require any inbound ports to be open. However, you might want to add an SSH rule so you can log into the container instance and examine the tasks with Docker commands. You can also add rules for HTTP and HTTPS if you want your container instance to host a task that runs a web server. Container instances do require external network access to communicate with the Amazon ECS service endpoint. Complete the following steps to add these optional security group rules.

Add the following three inbound rules to your security group.For information about how to create a security group, see Configure security group rules in the *Amazon EC2 User Guide*.

Option	Value
HTTP rule	Type: HTTP Source: Anywhere (`0.0.0.0/0`) This option automatically adds the 0.0.0.0/0 IPv4 CIDR block as the source. This is acceptable for a short time in a test environment, but it's unsafe in production environments. In production, authorize only a specific IP address or range of addresses to access your instance.
HTTPS rule	Type: HTTPS

Option	Value	
	Source: Anywhere (`0.0.0.0/0`) This is acceptable for a short time in a test environment, but it's unsafe in production environments. In production, authorize only a specific IP address or range of addresses to access your instance.	

Option	Value	
SSH rule	Type: SSH Source: Custom, specify the public IP address of your computer or network in CIDR notation. To specify an individual IP address in CIDR notation, add the routing prefix `/32`. For example, if your IP address is `203.0.113.25`, specify `203.0.113.25/32`. If your company allocates addresses from a range, specify the entire range, such as `203.0.113.0/24`. **Important** For security reasons, we don't recommend that you allow SSH access from all IP addresses (`0.0.0.0/0`) to your instance, except for testing purposes and only for a short time.	

Create the credentials to connect to your EC2 instance

For Amazon ECS, a key pair is only needed if you intend on using the EC2 launch type.

AWS uses public-key cryptography to secure the login information for your instance. A Linux instance, such as an Amazon ECS container instance, has no password to use for SSH access. You use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your container instance, then provide the private key when you log in using SSH.

If you haven't created a key pair already, you can create one using the Amazon EC2 console. If you plan to launch instances in multiple regions, you'll need to create a key pair in each region. For more information about regions, see Regions and Availability Zones in the *Amazon EC2 User Guide*.

To create a key pair

- Use the Amazon EC2 console to create a key pair. For more information about creating a key pair, see Create a key pair in the *Amazon EC2 User Guide*.

For information about how to connect to your instance, see Connect to your Linux instance in the *Amazon EC2 User Guide*.

Install the AWS CLI

The AWS Management Console can be used to manage all operations manually with Amazon ECS. However, you can install the AWS CLI on your local desktop or a developer box so that you can build scripts that can automate common management tasks in Amazon ECS.

To use the AWS CLI with Amazon ECS, install the latest AWS CLI version. For information about installing the AWS CLI or upgrading it to the latest version, see Installing or updating to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

The AWS Command Line Interface (AWS CLI) is a unified tool that you can use to manage your AWS services. With this one tool alone, you can both control multiple AWS services and automate these services through scripts. The Amazon ECS commands in the AWS CLI are a reflection of the Amazon ECS API.

The AWS CLI is suitable for customers who prefer and are used to scripting and interfacing with a command line tool and know exactly which actions they want to perform on their Amazon ECS resources. The AWS CLI is also helpful to customers who want to familiarize themselves with the Amazon ECS APIs. Customers can use the AWS CLI to perform a number of operations on Amazon ECS resources, including Create, Read, Update, and Delete operations, directly from the command line interface.

Use the AWS CLI if you are or want to become familiar with the Amazon ECS APIs and corresponding CLI commands and want to write automated scripts and perform specific actions on Amazon ECS resources.

AWS also provides the command line tools AWS Tools for Windows PowerShell. For more information, see the AWS Tools for Windows PowerShell User Guide.

Next steps for using Amazon ECS

After installing the AWS CLI, there are many different tools you can utilize as you continue to use Amazon ECS. The following links explain what some of those tools are and give examples of how to use them with Amazon ECS.

- Create your first container image with Docker and push it to Amazon ECR for use in your Amazon ECS task definitions.
- Learn how to create an Amazon ECS Linux task for the Fargate launch type.
- Learn how to create an Amazon ECS Windows task for the Fargate launch type.
- Learn how to create an Amazon ECS Windows task for the EC2 launch type.
- Using your preferred programming language, define infrastructure or architecture as code with the Creating Amazon ECS resources using the AWS CDK.
- Define and manage all AWS resources in your environment with automated deployment using Using Amazon ECS with AWS CloudFormation.
- Use the complete Creating Amazon ECS resources using the AWS Copilot command line interface end-to-end developer workflow to create, release, and operate container applications that comply with AWS best practices for infrastructure.

Creating a container image for use on Amazon ECS

Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers.

Amazon ECS schedules containerized applications on to container instances or on to AWS Fargate. Containerized applications are packaged as container images. This example creates a container image for a web server.

You can create your first Docker image, and then push that image to Amazon ECR, which is a container registry, for use in your Amazon ECS task definitions. This walkthrough assumes that you possess a basic understanding of what Docker is and how it works. For more information about Docker, see What is Docker? and the Docker documentation.

Prerequisites

Before you begin, ensure the following prerequisites are met.

- Ensure you have completed the Amazon ECR setup steps. For more information, see Moving an image through its lifecycle in Amazon ECR in the *Amazon Elastic Container Registry User Guide*.
- Your user has the required IAM permissions to access and use the Amazon ECR service. For more information, see Amazon ECR managed policies.
- You have Docker installed. For Docker installation steps for Amazon Linux 2023, see Installing Docker on AL2023. For all other operating systems, see the Docker documentation at Docker Desktop overview.
- You have the AWS CLI installed and configured. For more information, see Installing or updating to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

If you don't have or need a local development environment and you prefer to use an Amazon EC2 instance to use Docker, we provide the following steps to launch an Amazon EC2 instance using Amazon Linux 2023 and install Docker Engine and the Docker CLI.

Installing Docker on AL2023

Docker is available on many different operating systems, including most modern Linux distributions, like Ubuntu, and even macOS and Windows. For more information about how to install Docker on your particular operating system, go to the Docker installation guide.

You do not need a local development system to use Docker. If you are using Amazon EC2 already, you can launch an Amazon Linux 2023 instance and install Docker to get started.

If you already have Docker installed, skip to Create a Docker image.

To install Docker on an Amazon EC2 instance using an Amazon Linux 2023 AMI

1. Launch an instance with the latest Amazon Linux 2023 AMI. For more information, see Launch an EC2 instance using the launch instance wizard in the console in the *Amazon EC2 User Guide*.
2. Connect to your instance. For more information, see Connect to your EC2 instance in the *Amazon EC2 User Guide*.
3. Update the installed packages and package cache on your instance.

```
sudo yum update -y
```

4. Install the most recent Docker Community Edition package.

```
sudo yum install docker
```

5. Start the Docker service.

```
sudo service docker start
```

6. Add the `ec2-user` to the `docker` group so you can execute Docker commands without using `sudo`.

```
sudo usermod -a -G docker ec2-user
```

7. Log out and log back in again to pick up the new `docker` group permissions. You can accomplish this by closing your current SSH terminal window and reconnecting to your instance in a new one. Your new SSH session will have the appropriate `docker` group permissions.

8. Verify that the `ec2-user` can run Docker commands without `sudo`.

```
docker info
```

Note

In some cases, you may need to reboot your instance to provide permissions for the `ec2-user` to access the Docker daemon. Try rebooting your instance if you see the following error:

```
Cannot connect to the Docker daemon. Is the docker daemon running on this
 host?
```

Create a Docker image

Amazon ECS task definitions use container images to launch containers on the container instances in your clusters. In this section, you create a Docker image of a simple web application, and test it on your local system or Amazon EC2 instance, and then push the image to the Amazon ECR container registry so you can use it in an Amazon ECS task definition.

To create a Docker image of a simple web application

1. Create a file called `Dockerfile`. A Dockerfile is a manifest that describes the base image to use for your Docker image and what you want installed and running on it. For more information about Dockerfiles, go to the Dockerfile Reference.

   ```
   touch Dockerfile
   ```

2. Edit the `Dockerfile` you just created and add the following content.

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest

   # Update installed packages and install Apache
   RUN yum update -y && \
    yum install -y httpd

   # Write hello world message
   RUN echo 'Hello World!' > /var/www/html/index.html

   # Configure Apache
   RUN echo 'mkdir -p /var/run/httpd' >> /root/run_apache.sh && \
    echo 'mkdir -p /var/lock/httpd' >> /root/run_apache.sh && \
    echo '/usr/sbin/httpd -D FOREGROUND' >> /root/run_apache.sh && \
    chmod 755 /root/run_apache.sh

   EXPOSE 80

   CMD /root/run_apache.sh
   ```

   This Dockerfile uses the public Amazon Linux 2023 image hosted on Amazon ECR Public. The RUN instructions update the package caches, installs some software packages for the web server, and then write the "Hello World!" content to the web servers document root. The `EXPOSE` instruction means that port 80 on the container is the one that is listening, and the `CMD` instruction starts the web server.

3. Build the Docker image from your Dockerfile.

> **Note**
>
> Some versions of Docker may require the full path to your Dockerfile in the following command, instead of the relative path shown below.
> If you run the command an ARM based system, such as Apple Silicon, use the --platform option "--platform linux/amd64".

```
docker build -t hello-world .
```

4. List your container image.

```
docker images --filter reference=hello-world
```

Output:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
 SIZE
hello-world         latest              e9ffedc8c286        4 minutes ago
 194MB
```

5. Run the newly built image. The `-p 80:80` option maps the exposed port 80 on the container to port 80 on the host system.

```
docker run -t -i -p 80:80 hello-world
```

> **Note**
>
> Output from the Apache web server is displayed in the terminal window. You can ignore the "`Could not reliably determine the fully qualified domain name`" message.

6. Open a browser and point to the server that is running Docker and hosting your container.

   - If you are using an EC2 instance, this is the **Public DNS** value for the server, which is the same address you use to connect to the instance with SSH. Make sure that the security group for your instance allows inbound traffic on port 80.

- If you are running Docker locally, point your browser to http://localhost/.

  You should see a web page with your "Hello World!" statement.

7. Stop the Docker container by typing **Ctrl + c**.

Push your image to Amazon Elastic Container Registry

Amazon ECR is a managed AWS managed image registry service. You can use the Docker CLI to push, pull, and manage images in your Amazon ECR repositories. For Amazon ECR product details, featured customer case studies, and FAQs, see the Amazon Elastic Container Registry product detail pages.

To tag your image and push it to Amazon ECR

1. Create an Amazon ECR repository to store your `hello-world` image. Note the `repositoryUri` in the output.

   Substitute `region`, with your AWS Region, for example, `us-east-1`.

   ```
   aws ecr create-repository --repository-name hello-repository --region region
   ```

   Output:

   ```
   {
       "repository": {
           "registryId": "aws_account_id",
           "repositoryName": "hello-repository",
           "repositoryArn": "arn:aws:ecr:region:aws_account_id:repository/hello-
   repository",
           "createdAt": 1505337806.0,
           "repositoryUri": "aws_account_id.dkr.ecr.region.amazonaws.com/hello-
   repository"
       }
   }
   ```

2. Tag the `hello-world` image with the `repositoryUri` value from the previous step.

   ```
   docker tag hello-world aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
   ```

3. Run the **aws ecr get-login-password** command. Specify the registry URI you want to authenticate to. For more information, see Registry Authentication in the *Amazon Elastic Container Registry User Guide*.

```
aws ecr get-login-password --region region | docker login --username AWS --
password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

Output:

```
Login Succeeded
```

> **Important**
>
> If you receive an error, install or upgrade to the latest version of the AWS CLI. For more information, see Installing or updating to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

4. Push the image to Amazon ECR with the `repositoryUri` value from the earlier step.

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/hello-repository
```

Clean up

To continue on with creating an Amazon ECS task definition and launching a task with your container image, skip to the Next steps. When you are done experimenting with your Amazon ECR image, you can delete the repository so you are not charged for image storage.

```
aws ecr delete-repository --repository-name hello-repository --region region --force
```

Next steps

Your task definitions require a task execution role. For more information, see Amazon ECS task execution IAM role.

After you have created and pushed your container image to Amazon ECR, you can use that image in a task definition. For more information, see one of the following:

- the section called “Learn how to create a Linux task for the Fargate launch type”

- the section called “Learn how to create a Windows task for the Fargate launch type”
- Creating an Amazon ECS Linux task for the Fargate launch type with the AWS CLI

Learn how to create an Amazon ECS Linux task for the Fargate launch type

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage your containers. You can host your containers on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks on AWS Fargate. For more information on Fargate, see AWS Fargate for Amazon ECS.

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks in the Regions where Amazon ECS supports AWS Fargate.

Complete the following steps to get started with Amazon ECS on AWS Fargate.

Prerequisites

Before you begin, complete the steps in Set up to use Amazon ECS and that your IAM user has the permissions specified in the `AdministratorAccess` IAM policy example.

The console attempts to automatically create the task execution IAM role, which is required for Fargate tasks. To ensure that the console is able to create this IAM role, one of the following must be true:

- Your user has administrator access. For more information, see Set up to use Amazon ECS.
- Your user has the IAM permissions to create a service role. For more information, see Creating a Role to Delegate Permissions to an AWS Service.
- A user with administrator access has manually created the task execution role so that it is available on the account to be used. For more information, see Amazon ECS task execution IAM role.

> **Important**
>
> The security group you select when creating a service with your task definition must have port 80 open for inbound traffic. Add the following inbound rule to your security group.

For information about how to create a security group, see Create a security group for your Amazon EC2 instance in the *Amazon EC2 User Guide*.

- Type: HTTP
- Protocol: TCP
- Port range: 80
- Source: Anywhere (`0.0.0.0/0`)

Step 1: Create the cluster

Create a cluster that uses the default VPC.

Before you begin, assign the appropriate IAM permission. For more information, see the section called “Amazon ECS cluster examples”.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, for **Cluster name**, enter a unique name.

   The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.
6. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
7. (Optional) To help identify your cluster, expand **Tags**, and then configure your tags.

   [Add a tag] Choose **Add tag** and do the following:

   - For **Key**, enter the key name.
   - For **Value**, enter the key value.

   [Remove a tag] Choose **Remove** to the right of the tag’s Key and Value.
8. Choose **Create**.

Step 2: Create a task definition

A task definition is like a blueprint for your application. Each time you launch a task in Amazon ECS, you specify a task definition. The service then knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container.

1. In the navigation pane, choose **Task Definitions**.
2. Choose **Create new Task Definition**, **Create new revision with JSON**.
3. Copy and paste the following example task definition into the box and then choose **Save**.

```
{
    "family": "sample-fargate",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
  "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
```

```
    "cpu": "256",
    "memory": "512"
}
```

4. Choose **Create**.

Step 3: Create the service

Create a service using the task definition.

1. In the navigation pane, choose **Clusters**, and then select the cluster you created in Step 1: Create the cluster.
2. From the **Services** tab, choose **Create**.
3. Under **Deployment configuration**, specify how your application is deployed.

   a. For **Task definition**, choose the task definition you created in Step 2: Create a task definition.

   b. For **Service name**, enter a name for your service.

   c. For **Desired tasks**, enter **1**.
4. Under **Networking**, you can create a new security group or choose an existing security group for your task. Ensure that the security group you use has the inbound rule listed under Prerequisites.
5. Choose **Create**.

Step 4: View your service

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you ran the service.
4. In the **Services** tab, under **Service name**, choose the service you created in Step 3: Create the service.
5. Choose the **Tasks** tab, and then choose the task in your service.
6. On the task page, in the **Configuration** section, under **Public IP**, choose **Open address**.

Step 5: Clean up

When you are finished using an Amazon ECS cluster, you should clean up the resources associated with it to avoid incurring charges for resources that you are not using.

Some Amazon ECS resources, such as tasks, services, clusters, and container instances, are cleaned up using the Amazon ECS console. Other resources, such as Amazon EC2 instances, Elastic Load Balancing load balancers, and Auto Scaling groups, must be cleaned up manually in the Amazon EC2 console or by deleting the AWS CloudFormation stack that created them.

1. In the navigation pane, choose **Clusters**.
2. On the **Clusters** page, select the cluster you created for this tutorial.
3. Choose the **Services** tab.
4. Select the service, and then choose **Delete**.
5. At the confirmation prompt, enter **delete** and then choose **Delete**. Alternatively, you can use the `Force delete` option to have Amazon ECS scale the service down on your behalf before deleting it.

   Wait until the service is deleted.
6. Choose **Delete Cluster**. At the confirmation prompt, enter **delete** ***`cluster-name`***, and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Learn how to create an Amazon ECS Windows task for the Fargate launch type

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks in the Regions where Amazon ECS supports AWS Fargate.

Complete the following steps to get started with Amazon ECS on AWS Fargate.

Prerequisites

Before you begin, complete the steps in Set up to use Amazon ECS and that your IAM user has the permissions specified in the `AdministratorAccess` IAM policy example.

The console attempts to automatically create the task execution IAM role, which is required for Fargate tasks. To ensure that the console is able to create this IAM role, one of the following must be true:

- Your user has administrator access. For more information, see Set up to use Amazon ECS.
- Your user has the IAM permissions to create a service role. For more information, see Creating a Role to Delegate Permissions to an AWS Service.
- A user with administrator access has manually created the task execution role so that it is available on the account to be used. For more information, see Amazon ECS task execution IAM role.

Important

The security group you select when creating a service with your task definition must have port 80 open for inbound traffic. Add the following inbound rule to your security group. For information about how to create a security group, see Create a security group for your Amazon EC2 instance in the *Amazon EC2 User Guide*.

- Type: HTTP
- Protocol: TCP
- Port range: 80
- Source: Anywhere (`0.0.0.0/0`)

Step 1: Create a cluster

You can create a new cluster called **windows** that uses the default VPC.

To create a cluster with the AWS Management Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, for **Cluster name**, enter **windows**.

6. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
7. (Optional) To help identify your cluster, expand **Tags**, and then configure your tags.

   [Add a tag] Choose **Add tag** and do the following:

   - For **Key**, enter the key name.
   - For **Value**, enter the key value.

   [Remove a tag] Choose **Remove** to the right of the tag's Key and Value.
8. Choose **Create**.

Step 2: Register a Windows task definition

Before you can run Windows containers in your Amazon ECS cluster, you must register a task definition. The following task definition example displays a simple webpage on port 8080 of a container instance with the `mcr.microsoft.com/windows/servercore/iis` container image.

To register the sample task definition with the AWS Management Console

1. In the navigation pane, choose **Task definitions**.
2. Choose **Create new task definition**, **Create new task definition with JSON**.
3. Copy and paste the following example task definition into the box and then choose **Save**.

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html
 -Type file -Value '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
```

```
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

4. Verify your information and choose **Create**.

Step 3: Create a service with your task definition

After you have registered your task definition, you can place tasks in your cluster with it. The following procedure creates a service with your task definition and places one task in your cluster.

To create a service from your task definition with the console

1. In the navigation pane, choose **Clusters**, and then select the cluster you created in Step 1: Create a cluster.
2. From the **Services** tab, choose **Create**.
3. Under **Deployment configuration**, specify how your application is deployed.
    a. For **Task definition**, choose the task definition you created in Step 2: Register a Windows task definition.
    b. For **Service name**, enter a name for your service.
    c. For **Desired tasks**, enter **1**.

4. Under **Networking**, you can create a security group or choose an existing group. Ensure that the security group you use has the inbound rule listed under Prerequisites.
5. Choose **Create**.

Step 4: View your service

After your service has launched a task into your cluster, you can view the service and open the IIS test page in a browser to verify that the container is running.

> **Note**
>
> It can take up to 15 minutes for your container instance to download and extract the Windows container base layers.

To view your service

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you ran the service.
4. In the **Services** tab, under **Service name**, choose the service you created in Step 3: Create a service with your task definition.
5. Choose the **Tasks** tab, and then choose the task in your service.
6. On the task page, in the **Configuration** section, under **Public IP**, choose **Open address**.

Step 5: Clean Up

When you are finished using an Amazon ECS cluster, you should clean up the resources associated with it to avoid incurring charges for resources that you are not using.

Some Amazon ECS resources, such as tasks, services, clusters, and container instances, are cleaned up using the Amazon ECS console. Other resources, such as Amazon EC2 instances, Elastic Load Balancing load balancers, and Auto Scaling groups, must be cleaned up manually in the Amazon EC2 console or by deleting the AWS CloudFormation stack that created them.

1. In the navigation pane, choose **Clusters**.

2. On the **Clusters** page, select the cluster you created for this tutorial.
3. Choose the **Services** tab.
4. Select the service, and then choose **Delete**.
5. At the confirmation prompt, enter **delete** and then choose **Delete**.

   Wait until the service is deleted.
6. Choose **Delete Cluster**. At the confirmation prompt, enter **delete** ***`cluster-name`***, and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Learn how to create an Amazon ECS Windows task for the EC2 launch type

Get started with Amazon ECS using the EC2 launch type by registering a task definition, creating a cluster, and creating a service in the console.

Complete the following steps to get started with Amazon ECS using the EC2 launch type.

Prerequisites

Before you begin, complete the steps in Set up to use Amazon ECS and that your IAM user has the permissions specified in the `AdministratorAccess` IAM policy example.

The console attempts to automatically create the task execution IAM role, which is required for Fargate tasks. To ensure that the console is able to create this IAM role, one of the following must be true:

- Your user has administrator access. For more information, see Set up to use Amazon ECS.
- Your user has the IAM permissions to create a service role. For more information, see Creating a Role to Delegate Permissions to an AWS Service.
- A user with administrator access has manually created the task execution role so that it is available on the account to be used. For more information, see Amazon ECS task execution IAM role.

Important

The security group you select when creating a service with your task definition must have port 80 open for inbound traffic. Add the following inbound rule to your security group. For information about how to create a security group, see Create a security group for your Amazon EC2 instance in the *Amazon EC2 User Guide*.

- Type: HTTP
- Protocol: TCP
- Port range: 80
- Source: Anywhere (`0.0.0.0/0`)

Step 1: Create a cluster

An Amazon ECS cluster is a logical grouping of tasks, services, and container instances.

The following steps walk you through creating a cluster with one Amazon EC2 instance registered to it which will enable us to run a task on it. If a specific field is not mentioned, leave the default console values.

To create a new cluster (Amazon ECS console)

Before you begin, assign the appropriate IAM permission. For more information, see the section called “Amazon ECS cluster examples”.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, for **Cluster name**, enter a unique name.

   The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.
6. (Optional) To change the VPC and subnets where your tasks and services launch, under **Networking**, perform any of the following operations:

   - To remove a subnet, under **Subnets**, choose **X** for each subnet that you want to remove.

- To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, select each subnet.

7. To add Amazon EC2 instances to your cluster, expand **Infrastructure**, and then select **Amazon EC2 instances**. Next, configure the Auto Scaling group which acts as the capacity provider:

   a. To using an existing Auto Scaling group, from **Auto Scaling group (ASG)**, select the group.

   b. To create a Auto Scaling group, from **Auto Scaling group (ASG)**, select **Create new group**, and then provide the following details about the group:

      - For **Operating system/Architecture**, choose the Amazon ECS-optimized AMI for the Auto Scaling group instances.
      - For **EC2 instance type**, choose the instance type for your workloads. For more information about the different instance types, see [Amazon EC2 Instances](#).

        Managed scaling works best if your Auto Scaling group uses the same or similar instance types.
      - For **SSH key pair**, choose the pair that proves your identity when you connect to the instance.
      - For **Capacity**, enter the minimum number and the maximum number of instances to launch in the Auto Scaling group. Amazon EC2 instances incur costs while they exist in your AWS resources. For more information, see [Amazon EC2 Pricing](#).

8. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.

9. (Optional) To manage the cluster tags, expand **Tags**, and then perform one of the following operations:

   [Add a tag] Choose **Add tag** and do the following:

   - For **Key**, enter the key name.
   - For **Value**, enter the key value.

   [Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

10. Choose **Create**.

Step 2: Register a task definition

To register the sample task definition with the AWS Management Console

1. In the navigation pane, choose **Task Definitions**.
2. Choose **Create new task definition**, **Create new task definition with JSON**.
3. Copy and paste the following example task definition into the box, and then choose **Save**.

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html
 -Type file -Value '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 443,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["EC2"]
```

```
}
```

4. Verify your information and choose **Create**.

Step 3: Create a Service

An Amazon ECS service helps you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it in order to maintain the desired number of tasks in the service. For more information on services, see Amazon ECS services.

To create a service

1. In the navigation pane, choose **Clusters**.
2. Select the cluster you created in Step 1: Create a cluster.
3. On the **Services** tab, choose **Create**.
4. In the **Environment** section, do the following:

   a. For **Compute options**, choose Launch type.

   b. For **Launch type**, select **EC2**
5. In the **Deployment configuration** section, do the following:

   a. For **Family**, choose the task definition you created in Step 2: Register a task definition.

   b. For **Service name**, enter a name for your service.

   c. For **Desired tasks**, enter **1**.
6. Review the options and choose **Create**.
7. Choose **View service** to review your service.

Step 4: View your Service

The service is a web-based application so you can view its containers with a web browser.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. Choose the cluster where you ran the service.

4. In the **Services** tab, under **Service name**, choose the service you created in Step 3: Create a Service.
5. Choose the **Tasks** tab, and then choose the task in your service.
6. On the task page, in the **Configuration** section, under **Public IP**, choose **Open address**. The screen shot below is the expected output.

Step 5: Clean Up

When you are finished using an Amazon ECS cluster, you should clean up the resources associated with it to avoid incurring charges for resources that you are not using.

Some Amazon ECS resources, such as tasks, services, clusters, and container instances, are cleaned up using the Amazon ECS console. Other resources, such as Amazon EC2 instances, Elastic Load Balancing load balancers, and Auto Scaling groups, must be cleaned up manually in the Amazon EC2 console or by deleting the AWS CloudFormation stack that created them.

1. In the navigation pane, choose **Clusters**.
2. On the **Clusters** page, select the cluster cluster you created for this tutorial.
3. Choose the **Services** tab.
4. Select the service, and then choose **Delete**.
5. At the confirmation prompt, enter **delete** and then choose **Delete**.

   Wait until the service is deleted.
6. Choose **Delete Cluster**. At the confirmation prompt, enter **delete** ***`cluster-name`***, and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Creating Amazon ECS resources using the AWS CDK

The AWS Cloud Development Kit (AWS CDK) is an Infrastructure-as-Code (IAC) framework that you can use to define AWS cloud infrastructure by using a programming language of your choosing. To define your own cloud infrastructure, you first write an app (in one of the CDK's supported languages) that contains one or more stacks. Then, you synthesize it to an AWS CloudFormation template and deploy your resources to your AWS account. Follow the steps in this topic to deploy a containerized web server with Amazon Elastic Container Service (Amazon ECS) and the AWS CDK on Fargate.

The AWS Construct Library, included with the CDK, provides modules that you can use to model the resources that AWS services provide. For popular services, the library provides curated constructs with smart defaults and best practices. One of these modules, specifically `aws-ecs-patterns`, provides high-level abstractions that you can use to define your containerized service and all the necessary supporting resources in a few lines of code.

This topic uses the `ApplicationLoadBalancedFargateService` construct. This construct deploys an Amazon ECS service on Fargate behind an application load balancer. The `aws-ecs-patterns` module also includes constructs that use a network load balancer and run on Amazon EC2.

Before starting this task, set up your AWS CDK development environment, and install the AWS CDK by running the following command. For instructions on how to set up your AWS CDK development environment, see Getting Started With the AWS CDK - Prerequisites.

```
npm install -g aws-cdk
```

Note

These instructions assume you are using AWS CDK v2.

Topics

- Step 1: Set up your AWS CDK project
- Step 2: Use the AWS CDK to define a containerized web server on Fargate
- Step 3: Test the web server
- Step 4: Clean up

- Next steps

Step 1: Set up your AWS CDK project

Create a directory for your new AWS CDK app and initialize the project.

TypeScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language typescript
```

JavaScript

```
mkdir hello-ecs
cd hello-ecs
cdk init --language javascript
```

Python

```
mkdir hello-ecs
cd hello-ecs
cdk init --language python
```

After the project is started, activate the project's virtual environment and install the AWS CDK's baseline dependencies.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir hello-ecs
cd hello-ecs
cdk init --language java
```

Import this Maven project to your Java IDE. For example, in Eclipse, use **File** > **Import** > **Maven** > **Existing Maven Projects**.

C#

```
mkdir hello-ecs
cd hello-ecs
cdk init --language csharp
```

Go

```
mkdir hello-ecs
cd hello-ecs
cdk init --language go
```

> **Note**
>
> The AWS CDK application template uses the name of the project directory to generate names for source files and classes. In this example, the directory is named `hello-ecs`. If you use a different project directory name, your app won't match these instructions.

AWS CDK v2 includes stable constructs for all AWS services in a single package that's called `aws-cdk-lib`. This package is installed as a dependency when you initialize the project. When working with certain programming languages, the package is installed when you build the project for the first time. This topic covers how to use an Amazon ECS Patterns construct, which provides high-level abstractions for working with Amazon ECS. This module relies on Amazon ECS constructs and other constructs to provision the resources that your Amazon ECS application needs.

The names that you use to import these libraries into your CDK application might differ slightly depending on which programming language you use. For reference, the following are the names that are used in each supported CDK programming language.

TypeScript

```
aws-cdk-lib/aws-ecs
aws-cdk-lib/aws-ecs-patterns
```

JavaScript

```
aws-cdk-lib/aws-ecs
```

```
aws-cdk-lib/aws-ecs-patterns
```

Python

```
aws_cdk.aws_ecs
aws_cdk.aws_ecs_patterns
```

Java

```
software.amazon.awscdk.services.ecs
software.amazon.awscdk.services.ecs.patterns
```

C#

```
Amazon.CDK.AWS.ECS
Amazon.CDK.AWS.ECS.Patterns
```

Go

```
github.com/aws/aws-cdk-go/awscdk/v2/awsecs
github.com/aws/aws-cdk-go/awscdk/v2/awsecspatterns
```

Step 2: Use the AWS CDK to define a containerized web server on Fargate

Use the container image `amazon-ecs-sample`. This image contains a PHP web app that runs on Nginx.

In the AWS CDK project that you created, edit the file that contains the stack definition to resemble one of the following examples.

> **Note**
>
> A stack is a unit of deployment. All resources must be in a stack, and all the resources that are in a stack are deployed at the same time. If a resource fails to deploy, any other resources that were already deployed are rolled back. An AWS CDK app can contain multiple stacks, and resources in one stack can refer to resources in another stack.

TypeScript

Update `lib/hello-ecs-stack.ts` so that it resembles the following.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as ecsp from 'aws-cdk-lib/aws-ecs-patterns';

export class HelloEcsStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
      taskImageOptions: {
        image: ecs.ContainerImage.fromRegistry('public.ecr.aws/ecs-sample-image/
amazon-ecs-sample:latest'),
      },
      publicLoadBalancer: true
    });
  }
}
```

JavaScript

Update `lib/hello-ecs-stack.js` so that it resembles the following.

```
const cdk = require('aws-cdk-lib');
const { Construct } = require('constructs');
const ecs = require('aws-cdk-lib/aws-ecs');
const ecsp = require('aws-cdk-lib/aws-ecs-patterns');

class HelloEcsStack extends cdk.Stack {
  constructor(scope = Construct, id = string, props = cdk.StackProps) {
    super(scope, id, props);

    new ecsp.ApplicationLoadBalancedFargateService(this, 'MyWebServer', {
      taskImageOptions: {
        image: ecs.ContainerImage.fromRegistry('amazon/amazon-ecs-sample'),
      },
      publicLoadBalancer: true
    });
  }
```

```
}

module.exports = { HelloEcsStack }
```

Python

Update `hello-ecs/hello_ecs_stack.py` so that it resembles the following.

```
import aws_cdk as cdk
from constructs import Construct

import aws_cdk.aws_ecs as ecs
import aws_cdk.aws_ecs_patterns as ecsp

class HelloEcsStack(cdk.Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ecsp.ApplicationLoadBalancedFargateService(self, "MyWebServer",
            task_image_options=ecsp.ApplicationLoadBalancedTaskImageOptions(
                image=ecs.ContainerImage.from_registry("amazon/amazon-ecs-sample")),
            public_load_balancer=True
        )
```

Java

Update `src/main/java/com.myorg/HelloEcsStack.java` so that it resembles the following.

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;

import software.amazon.awscdk.services.ecs.ContainerImage;
import
 software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedFargateService;
import
 software.amazon.awscdk.services.ecs.patterns.ApplicationLoadBalancedTaskImageOptions;

public class HelloEcsStack extends Stack {
```

```
    public HelloEcsStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloEcsStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        ApplicationLoadBalancedFargateService.Builder.create(this, "MyWebServer")
          .taskImageOptions(ApplicationLoadBalancedTaskImageOptions.builder()
            .image(ContainerImage.fromRegistry("amazon/amazon-ecs-sample"))
            .build())
          .publicLoadBalancer(true)
          .build();
    }
}
```

C#

Update `src/HelloEcs/HelloEcsStack.cs` so that it resembles the following.

```
using Amazon.CDK;
using Constructs;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
namespace HelloEcs
{
    public class HelloEcsStack : Stack
    {
        internal HelloEcsStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            new ApplicationLoadBalancedFargateService(this, "MyWebServer",
                new ApplicationLoadBalancedFargateServiceProps
                {
                    TaskImageOptions = new ApplicationLoadBalancedTaskImageOptions
                    {
                        Image = ContainerImage.FromRegistry("amazon/amazon-ecs-
sample")
                    },
                    PublicLoadBalancer = true
                });
        }
    }
```

```
}
```

Go

Update `hello-ecs.go` so that it resembles the following.

```
package main

import (
 "github.com/aws/aws-cdk-go/awscdk/v2"
 // "github.com/aws/aws-cdk-go/awscdk/v2/awssqs"
 "github.com/aws/aws-cdk-go/awscdk/v2/awsecs"
 "github.com/aws/aws-cdk-go/awscdk/v2/awsecspatterns"
 "github.com/aws/constructs-go/constructs/v10"
 "github.com/aws/jsii-runtime-go"
)

type HelloEcsStackProps struct {
 awscdk.StackProps
}

func NewHelloEcsStack(scope constructs.Construct, id string, props
 *HelloEcsStackProps) awscdk.Stack {
 var sprops awscdk.StackProps
 if props != nil {
  sprops = props.StackProps
 }
 stack := awscdk.NewStack(scope, &id, &sprops)

 // The code that defines your stack goes here

 // example resource
 // queue := awssqs.NewQueue(stack, jsii.String("HelloEcsQueue"),
 &awssqs.QueueProps{
 //  VisibilityTimeout: awscdk.Duration_Seconds(jsii.Number(300)),
 // })
 res := awsecspatterns.NewApplicationLoadBalancedFargateService(stack,
 jsii.String("MyWebServer"),
  &awsecspatterns.ApplicationLoadBalancedFargateServiceProps{
   TaskImageOptions: &awsecspatterns.ApplicationLoadBalancedTaskImageOptions{
    Image: awsecs.ContainerImage_FromRegistry(jsii.String("amazon/amazon-ecs-
sample"), &awsecs.RepositoryImageProps{}),
   },
  },
```

```
 )
 awscdk.NewCfnOutput(stack, jsii.String("LoadBalancerDNS"),
 &awscdk.CfnOutputProps{Value: res.LoadBalancer().LoadBalancerDnsName()})

 return stack
}

func main() {
 defer jsii.Close()

 app := awscdk.NewApp(nil)

 NewHelloEcsStack(app, "HelloEcsStack", &HelloEcsStackProps{
  awscdk.StackProps{
   Env: env(),
  },
 })

 app.Synth(nil)
}

// env determines the AWS environment (account+region) in which our stack is to
// be deployed. For more information see: https://docs.aws.amazon.com/cdk/latest/
guide/environments.html
func env() *awscdk.Environment {
 // If unspecified, this stack will be "environment-agnostic".
 // Account/Region-dependent features and context lookups will not work, but a
 // single synthesized template can be deployed anywhere.
 //---------------------------------------------------------------------------
 return nil

 // Uncomment if you know exactly what account and region you want to deploy
 // the stack to. This is the recommendation for production stacks.
 //---------------------------------------------------------------------------
 // return &awscdk.Environment{
 //  Account: jsii.String("123456789012"),
 //  Region:  jsii.String("us-east-1"),
 // }

 // Uncomment to specialize this stack for the AWS Account and Region that are
 // implied by the current CLI configuration. This is recommended for dev
 // stacks.
 //---------------------------------------------------------------------------
 // return &awscdk.Environment{
```

```
    //  Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),
    //  Region:  jsii.String(os.Getenv("CDK_DEFAULT_REGION")),
    // }
   }
```

The preceding short snippet includes the following:

- The service's logical name: `MyWebServer`.
- The container image that was obtained from the Amazon ECR Public Gallery: `amazon/amazon-ecs-sample`.
- Other relevant information, such as the fact that the load balancer has a public address and is accessible from the Internet.

The AWS CDK will create all the resources that are required to deploy the web server including the following resources. These resources were omitted in this example.

- Amazon ECS cluster
- Amazon VPC and Amazon EC2 instances
- Auto Scaling group
- Application Load Balancer
- IAM roles and policies

Some automatically provisioned resources are shared by all Amazon ECS services defined in the stack.

Save the source file, then run the `cdk synth` command in your application's main directory. The AWS CDK runs the app and synthesizes an AWS CloudFormation template from it, and then displays the template. The template is an approximately 600-line YAML file. The beginning of the file is shown here. Your template might differ from this example.

```
Resources:
  MyWebServerLB3B5FD3AB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      LoadBalancerAttributes:
```

```
        - Key: deletion_protection.enabled
          Value: "false"
      Scheme: internet-facing
      SecurityGroups:
        - Fn::GetAtt:
            - MyWebServerLBSecurityGroup01B285AA
            - GroupId
      Subnets:
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1Subnet3C273B99
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2Subnet95FF715A
      Type: application
    DependsOn:
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1DefaultRouteFF4E2178
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2DefaultRouteB1375520
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/Resource
  MyWebServerLBSecurityGroup01B285AA:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Automatically created Security Group for ELB
 HelloEcsStackMyWebServerLB06757F57
      SecurityGroupIngress:
        - CidrIp: 0.0.0.0/0
          Description: Allow from anyone on port 80
          FromPort: 80
          IpProtocol: tcp
          ToPort: 80
      VpcId:
        Ref: EcsDefaultClusterMnL3mNNYNVpc7788A521
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/SecurityGroup/Resource
# and so on for another few hundred lines
```

To deploy the service in your AWS account, run the `cdk deploy` command in your application's main directory. You're asked to approve the IAM policies that the AWS CDK generated.

The deployment takes several minutes during which the AWS CDK creates several resources. The last few lines of the output from the deployment include the load balancer's public hostname and your new web server's URL. They are as follows.

```
Outputs:
HelloEcsStack.MyWebServerLoadBalancerDNSXXXXXXX = Hello-MyWeb-ZZZZZZZZZZZZZ-
ZZZZZZZZZZ.us-west-2.elb.amazonaws.com
```

```
HelloEcsStack.MyWebServerServiceURLYYYYYYYY = http://Hello-MyWeb-ZZZZZZZZZZZZZ-
ZZZZZZZZZZ.us-west-2.elb.amazonaws.com
```

Step 3: Test the web server

Copy the URL from the deployment output and paste it into your web browser. The following welcome message from the web server is displayed.

Step 4: Clean up

After you're finished with the web server, end the service using the CDK by running the `cdk destroy` command in your application's main directory. Doing this prevents you from incurring any unintended charges in the future.

Next steps

To learn more about how to develop AWS infrastructure using the AWS CDK, see the AWS CDK Developer Guide.

For information about writing AWS CDK apps in your language of choice, see the following:

TypeScript

  Working with the AWS CDK in TypeScript

JavaScript

  Working with the AWS CDK in JavaScript

Python

Working with the AWS CDK in Python

Java

Working with the AWS CDK in Java

C#

Working with the AWS CDK in C#

Go

Working with the AWS CDK in Go

For more information about the AWS Construct Library modules used in this topic, see the following AWS CDK API Reference overviews.

- aws-ecs
- aws-ecs-patterns

Creating Amazon ECS resources using the AWS Copilot command line interface

The AWS Copilot command line interface (CLI) commands simplify building, releasing, and operating production-ready containerized applications on Amazon ECS from a local development environment. The AWS Copilot CLI aligns with developer workflows that support modern application best practices: from using infrastructure as code to creating a CI/CD pipeline provisioned on behalf of a user. Use the AWS Copilot CLI as part of your everyday development and testing cycle as an alternative to the AWS Management Console.

AWS Copilot currently supports Linux, macOS, and Windows systems. For more information about the latest version of the AWS Copilot CLI, see Releases.

> **Note**
>
> The source code for the AWS Copilot CLI is available on GitHub. We recommend that you submit issues and pull requests for changes that you would like to have included. However, Amazon Web Services doesn't currently support running modified copies of AWS Copilot

code. Report issues with AWS Copilot by connecting with us on Gitter or GitHub where you can open issues, provide feedback, and report bugs.
For information about Copilot development, see What's the future of Copilot?.

Topics

- Installing the AWS Copilot CLI
- Deploying a sample Amazon ECS application using the AWS Copilot CLI

Additional documentation for the AWS Copilot CLI is available on the AWS Copilot website

Installing the AWS Copilot CLI

You can install the AWS Copilot CLI by using Homebrew or by manually downloading the binary with the following steps.

Use Homebrew

The following command is used to install the AWS Copilot CLI on your macOS or Linux system using Homebrew. Before installation, you should have Homebrew installed. For more information, see Homebrew.

```
brew install aws/tap/copilot-cli
```

Download binary

As an alternative to Homebrew, you can manually install the AWS Copilot CLI on your macOS, Windows, or Linux system. Use the following command for your operating system to download the binary. The macOS and Linux examples also include commands that apply execute permissions to the binary, and list the help menu to verify that the installation works.

macOS

For macOS:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-darwin \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

For macOS ARM systems:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-darwin-arm64 \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Linux

For Linux x86 (64-bit) systems:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-linux \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

For Linux ARM systems:

```
sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releases/
latest/download/copilot-linux-arm64 \
   && sudo chmod +x /usr/local/bin/copilot \
   && copilot --help
```

Windows

Using Powershell, run the following command:

```
New-Item -Path 'C:\copilot' -ItemType directory; `
  Invoke-WebRequest -OutFile 'C:\copilot\copilot.exe' https://github.com/aws/
copilot-cli/releases/latest/download/copilot-windows.exe
```

(Optional) Verify the manually installed AWS Copilot CLI using PGP signatures

The AWS Copilot CLI executables are cryptographically signed using PGP signatures. The PGP signatures can be used to verify the validity of the AWS Copilot CLI executable. Use the following steps to verify the signatures using the GnuPG tool.

1. Download and install GnuPG. For more information, see the GnuPG website.

macOS

We recommend using Homebrew. Install Homebrew using the instructions from their website. For more information, see Homebrew. After Homebrew is installed, use the following command from your macOS terminal.

```
brew install gnupg
```

Linux

Install gpg using the package manager on your flavor of Linux.

Windows

Download the Windows simple installer from the GnuPG website and install as an Administrator. After you install GnuPG, close and reopen the Administrator PowerShell.

For more information, see GnuPG Download.

2. Verify the GnuPG path is added to your environment path.

macOS

```
echo $PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
PATH=$PATH:<path to GnuPG executable files>
```

Linux

```
echo $PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
export PATH=$PATH:<path to GnuPG executable files>
```

Windows

```
Write-Output $Env:PATH
```

If you do not see the GnuPG path in the output, run the following command to add it to the path.

```
$Env:PATH += ";<path to GnuPG executable files>"
```

3. Create a local plain text file.

macOS

On the terminal, enter:

```
touch <public_key_filename.txt>
```

Open the file with TextEdit.

Linux

Create a text file in a text editor such as gedit. Save as `public_key_filename.txt`

Windows

Create a text file in a text editor such as Notepad. Save as `public_key_filename.txt`

4. Add the following contents of the Amazon ECS PGP public key and save the file.


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2fl74lmavr4Vg
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
N+6pHBJrKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO
2K7oVX8Oc3j5+ZilhL/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNIcO
lFTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn
5lBPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq
lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQq0osYXaQbvlnSWOoc1y/5Zb
```

```
zizHTJIhLtUyls9WisP2s0emeHZicVMfW61EgPrJAiupgc7kyZvFt4YwfwARAQAB
tCRBbWF6b24gRUNTIDxlY3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhwEEAECAAYF
AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPLt5bV1+rkqOlw+0gZ4Q/ue/YbWtOAt4qY0OcEo0HgcnaX
lsB827QIfZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmazA
McWB4HUMNrhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF
4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMZpVkUr+C252IaH1HZFEz+TvBVQM
Y8OWWxmIJW+J6evjo3N1eO19UHv71jvoF8zljbI4bsL2c+QTJmOv7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBBwNJzQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItenatK
lEJQhYtyVXcBlh6Yn/wzNg2NWOwb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPKmyGopsmN/reMuilPdINb249nA0dzoN+nj+tTFOYCIaLaFyjs
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX
gAAKCRC86dmkLVF4T9iFEACEnkm1dNXsWUx34R3c0vamHrPxvfkyI1FlEUen8D1h
uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZUgvXBL046JhG
eM17+crgUyQeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFtgAsc9rk+
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1
0X90J7cC1KOyhlEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlz3+6xBIm/qe
bFyLUnf4WoiuOplAaJhK9pRY+XEnGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr
KVHUq1TZD7cvMnnNKEELTUcKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5Un5zwqkwwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqRt9X42yF1PIEVRneBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+ekr5TptVszS9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAxNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMucIINtH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+JIhZwdD8Mx2K+LVVVu/aWkNrfMuNwyDUciSI4D5QHa8T+F8fgN4OTpwYjirzel
5yoICMr9hVcbzDNv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXRYtVvYInEhLVrJ0aV6zHFoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
t/MOdjXr1fjF4pThEi5u6WsuRdFwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
2iHiOKIpQqLbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QuKE+I
R6jA0frUNt2jhiGG/F8RceXzohaaC/Cx7LUCUFWc0n7z32C9/Dtj7I1PMOacdZzz
bjJzRKO/ZDv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXPO0EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIACQUCWrVJqwIbDAAKCRC86dmkLVF4T+ZdD/9x/8APzgNJF3o3STrF
jvnV1ycyhWYGAeBJiu7wjsNWwzMFOv15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
X7DR0JszaH9wrYTxZLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt
Rwe/uwdibI0CagEzyX+2D3kTOlHO5XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZnl3LSmZyE0EQehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmcliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBWDhBPVPrruLIaenTtZEOsPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
```

```
Gc5r9ysjkfH6+6rbi1ujxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUUW
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcW+jsY9IhbEzqN5yQYGi4pVmDkY5vu
lXbJnbqPKpRXgM9BecV9AMbPgbDq/5LnHJJXg+G8YQOgp4lR/hC1TEFdIp5wM8AK
CWsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAr77kj7j2QR2SZeOSlFBvV7oS
mFeSNnz9xZssqrsm6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0mOmptr+xLLsKcaP7ogIXsyZnrEAEsvW8PnfayoiPCdc3cMCR/lTnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4blD32WqvyxESo7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSGm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTssoG+3OOcGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikAOlvE3/wgMqCXscbycbLjLD/bXIuFWo3rzoezeXjgi/DJx
jKBAyBTYO5nMcth1O9oaFd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvf
KeM1rO8Jm3iRAc5a/D0AEQEAAYkEPgQYAQIACQUCWrVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bXO1dQR4tKhhQ
P0LRqy6zlBY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOclTb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTfloS
Of4inq71kjmGK+TlzQ6mUMQUg228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bhIIbQyEUB9gKCmUFaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphUl7T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLrWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA0OQxG
AHMOdJQQvBsmqMvuAdjkDWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vSjKpCNg3EIJSgqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8MelN9qr5KcKQLmfdfBNEyyceBhyVl0MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRyKToTjwOb73CGkBZZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivhoWwzjpfQKhwcU9lABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBXlR
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMii+mWra23EwjChaxpvjjcUH
5ilLc5Zq781aCYRygYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAX7kPMD3Lof4KldD
Q8ppQriUvxVo+4nPV6rpTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCEl7nbFrm0vIiA75Sa8KnywTDsyZsu3XcOcf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWHOis6bq8l9RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYlNr3lydh+dFHIEkH53HzQe6l88HEic
+0jVnLkCDQRa55wJARAAyLya2Lx6gyoWoJN1a6740q3o8e9d4KggQOfGMTCflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNJyjMNT1gAgrhPNB4HtXBXum2wS57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOIzQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSvOUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJIgjNvJscuUWb
```

```
+QEqJIRleJRhrO+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIxTn9xUc0UMKFeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm9lhkGwqTZgOlRX5eqPX+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6sYKqroYwBGgZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9zlW8qsA
EQEAAYkEPgQYAQgACQUCWuecCQIbAgIpCRC86dmkLVF4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykD/4tOLRHFHXuKUcxgGaubUcVtsFrwBKma1cYjqaPms8u
6Sk0wfGRI32G/GhOrp0Ts/MOkbObq6VLTh8N5Yc/53MEl8zQFw9Y5AmRoW4PZXER
ujs5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEFHYT8UTxwnGmXOvNkMM7md
xD3CV5q6VAte8WKBo/220II3fcQlc9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW/NN2gju2G3Lu/TlFUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUhQWeSNHI+fpe5oX6FALPT9JLDce8OZz1i
cZZ0MELP37mOOQun0AlmHm/hVzf0f311PtbzcqWaE51tJvgUR/nZFo6Ta3O5Ezhs
3VlEJNQ1Ijf/6DH87SxvAoRIARCuZd0qxBcDK0avpFzUtbJd24lRA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyz0WTDVmt
g0U1CO1ezokMSqkJ7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtfsCDvALCtqL0jduSkUo+RXcBItmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNtm9mw9ydIl1yjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6MlI2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxzlbNpS+zxt9r0MiLgcLyspWxSdmoYGZ6nQP
RO5Nm/ZVS+u2imPCRzNUZEMa+dlE6kHx0rS0dPiuJ4O7NtPeYDKkoQtNagspsDvh
cK7CSqAiKMq06UBTxqlTSRkm62eOCtcs3p3OeHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+45lcCfmcVt94TFNL5HwEUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNJ3bCLuc/jq7ysGq69xiKmTlUeXFm+aojcRO5i
zyShIRJZ0GZfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNxlf3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRGlU/LpNSefnvDFTtEIRcpOHc
bhayG0bk51Bd4mioOXnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj2O+X+J
qX2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==
=Gi5D
-----END PGP PUBLIC KEY BLOCK-----
```

The details of the Amazon ECS PGP public key for reference:

```
Key ID: BCE9D9A42D51784F
Type: RSA
Size: 4096/4096
Expires: Never
User ID: Amazon ECS
Key fingerprint: F34C 3DDA E729 26B0 79BE AEC6 BCE9 D9A4 2D51 784F
```

5. Import the file with the Amazon ECS PGP public key with the following command in the terminal.

```
gpg --import <public_key_filename.txt>
```

6. Download the AWS Copilot CLI signatures. The signatures are ASCII detached PGP signatures stored in files with the extension `.asc`. The signatures file has the same name as its corresponding executable, with `.asc` appended.

   macOS

   For macOS systems, run the following command.

   ```
   sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-darwin.asc
   ```

   Linux

   For Linux x86 (64-bit) systems, run the following command.

   ```
   sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux.asc
   ```

   For Linux ARM systems, run the following command.

   ```
   sudo curl -Lo copilot.asc https://github.com/aws/copilot-cli/releases/latest/download/copilot-linux-arm64.asc
   ```

   Windows

   Using Powershell, run the following command.

   ```
   Invoke-WebRequest -OutFile 'C:\copilot\copilot.asc' https://github.com/aws/copilot-cli/releases/latest/download/copilot-windows.exe.asc
   ```

7. Verify the signature with the following command.

   - For macOS and Linux systems:

     ```
     gpg --verify copilot.asc /usr/local/bin/copilot
     ```

   - For Windows systems:

```
gpg --verify 'C:\copilot\copilot.asc' 'C:\copilot\copilot.exe'
```

Expected output:

```
gpg: Signature made Tue Apr  3 13:29:30 2018 PDT
gpg:                using RSA key DE3CBD61ADAF8B8E
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: EB3D F841 E2C9 212A 2BD4  2232 DE3C BD61 ADAF 8B8E
```

Important

The warning in the output is expected and is not problematic. It occurs because there is not a chain of trust between your personal PGP key (if you have one) and the Amazon ECS PGP key. For more information, see Web of trust.

8. For Windows installations, run the following command on Powershell to add AWS Copilot directory to the path.

```
$Env:PATH += ";<path to Copilot executable files>"
```

Deploying a sample Amazon ECS application using the AWS Copilot CLI

After installing the AWS Copilot CLI, you can follow these steps to deploy a sample app, verify the deployment, and clean up resources.

Prerequisites

Before you begin, make sure that you meet the following prerequisites:

- Install and configure the AWS CLI. For more information, see AWS Command Line Interface.
- Run `aws configure` to set up a default profile that the AWS Copilot CLI will use to manage your application and services.
- Install and run Docker. For more information, see Get started with Docker.

Deploy a sample Amazon ECS application using a single command

1. Deploy a sample web application that is cloned from a GitHub repository using the following command. For more information about AWS Copilot `init` and its flags, see the [AWS Copilot documentation](https://aws.github.io/copilot-cli/docs/commands/init/).

```
git clone https://github.com/aws-samples/aws-copilot-sample-service.git demo-app && \
cd demo-app &&                                         \
copilot init --app demo                                \
  --name api                                           \
  --type 'Load Balanced Web Service'                   \
  --dockerfile './Dockerfile'                          \
  --port 80                                            \
  --tag  latest                                        \
  --deploy
```

2. After the deployment is complete, the AWS Copilot CLI will return a URL that you can use to verify the deployment. You can also use the following commands to verify the app's status.

   - List all of your AWS Copilot applications.

   ```
   copilot app ls
   ```

   - Show information about the environments and services in your application.

   ```
   copilot app show
   ```

   - Show information about your environments.

   ```
   copilot env ls
   ```

   - Show information about the service, including endpoints, capacity and related resources.

   ```
   copilot svc show
   ```

   - List of all the services in an application.

   ```
   copilot svc ls
   ```

   - Show logs of a deployed service.

```
copilot svc logs
```

- Show service status.

```
copilot svc status
```

3. When you're finished with this demo, run the following command to clean up associated resources and avoid incurring charges for unused resources.

```
copilot app delete
```

Using Amazon ECS with AWS CloudFormation

Amazon ECS is integrated with AWS CloudFormation, a service that you can use to model and set up AWS resources with templates that you define. AWS CloudFormation uses **templates** that are either a `YAML` or `JSON` formatted text file. Templates are like blueprints for the AWS resource you want to create. When you create and submit a template, AWS CloudFormation creates a **stack**. You manage the resources you defined in your template through the stack. When you want to create, update, or delete a resource, you create, update, or delete the stack that was created from that resource. When it comes to updating your stacks, you need to create a **change set** first. Change sets show you what is impacted by the change before you make it. This keeps you from deleting databases accidently by changing your database name, for example. For more information on templates, stacks, and change sets, see How AWS CloudFormation works in the *AWS CloudFormation User Guide*.

By using AWS CloudFormation, you can spend less time creating and managing your resources and infrastructure. You can create a template that describes all the AWS resources that you want, such as Amazon ECS clusters, task definitions, services. Then, AWS CloudFormation takes care of provisioning and configuring those resources for you.

AWS CloudFormation also allows you to reuse your template to set up your Amazon ECS resources in a consistent and repeatable manner. You describe your resources one time and then provision the same resources again across multiple AWS accounts and AWS Regions.

AWS CloudFormation templates can be used with both the AWS Management Console or the AWS Command Line Interface to create resources.

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation Command Line Interface User Guide

Topics

Creating Amazon ECS resources using the AWS CloudFormation console

One way to use Amazon ECS with AWS CloudFormation is through the AWS Management Console. Here you can create your AWS CloudFormation stacks for Amazon ECS components like task definitions, clusters, and services and deploy them directly from the console. The following tutorial shows how you can use the AWS CloudFormation console to create Amazon ECS resources using a template.

Prerequisites

This tutorial assumes that the following prerequisites have ben completed.

- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.

Step 1: Create a stack template

Use the following steps to create an AWS CloudFormation stack template for an Amazon ECS service and other related resources.

1. Using a text editor of your choice, create a file called `ecs-tutorial-template.yaml`.
2. In the `ecs-tutorial-template.yaml` file, paste the following template and save the changes.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] ECS: load-balanced-web-application'

Parameters:
  VpcCidr:
    Type: String
    Default: '10.0.0.0/16'
    Description: CIDR block for the VPC
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest'
    Description: Container image to use in task definition
```

```
  PublicSubnet1Cidr:
    Type: String
    Default: '10.0.1.0/24'
    Description: CIDR block for public subnet 1

  PublicSubnet2Cidr:
    Type: String
    Default: '10.0.2.0/24'
    Description: CIDR block for public subnet 2

  PrivateSubnet1Cidr:
    Type: String
    Default: '10.0.3.0/24'
    Description: CIDR block for private subnet 1

  PrivateSubnet2Cidr:
    Type: String
    Default: '10.0.4.0/24'
    Description: CIDR block for private subnet 2

  ServiceName:
    Type: String
    Default: 'tutorial-app'
    Description: Name of the ECS service

  ContainerPort:
    Type: Number
    Default: 80
    Description: Port on which the container listens

  DesiredCount:
    Type: Number
    Default: 2
    Description: Desired number of tasks

  MinCapacity:
    Type: Number
    Default: 1
    Description: Minimum number of tasks for auto scaling

  MaxCapacity:
    Type: Number
    Default: 10
    Description: Maximum number of tasks for auto scaling
```

```
Resources:
  # VPC and Networking
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-vpc'

  # Internet Gateway
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-igw'

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  # Public Subnets for ALB
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PublicSubnet1Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-1'

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
```

```
      CidrBlock: !Ref PublicSubnet2Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-2'

  # Private Subnets for ECS Tasks
  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet1Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-1'

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet2Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-2'

  # NAT Gateways for private subnet internet access
  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-1'

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
```

```
        Value: !Sub '${AWS::StackName}-nat-eip-2'

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-1'

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-2'

  # Route Tables
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-routes'

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
```

```
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-1'

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-2'

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
```

```
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  # Security Groups
  ALBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-alb-sg'
      GroupDescription: Security group for Application Load Balancer
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
          Description: Allow HTTP traffic from internet
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb-sg'

  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-ecs-sg'
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: !Ref ContainerPort
          ToPort: !Ref ContainerPort
          SourceSecurityGroupId: !Ref ALBSecurityGroup
          Description: Allow traffic from ALB
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-ecs-sg'
```

```
  # Application Load Balancer
  ApplicationLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: !Sub '${AWS::StackName}-alb'
      Scheme: internet-facing
      Type: application
      Subnets:
        - !Ref PublicSubnet1
        - !Ref PublicSubnet2
      SecurityGroups:
        - !Ref ALBSecurityGroup
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb'

  ALBTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: !Sub '${AWS::StackName}-tg'
      Port: !Ref ContainerPort
      Protocol: HTTP
      VpcId: !Ref VPC
      TargetType: ip
      HealthCheckIntervalSeconds: 30
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      UnhealthyThresholdCount: 5
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-tg'

  ALBListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref ALBTargetGroup
      LoadBalancerArn: !Ref ApplicationLoadBalancer
      Port: 80
      Protocol: HTTP
```

```
  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Sub '${AWS::StackName}-cluster'
      CapacityProviders:
        - FARGATE
        - FARGATE_SPOT
      DefaultCapacityProviderStrategy:
        - CapacityProvider: FARGATE
          Weight: 1
        - CapacityProvider: FARGATE_SPOT
          Weight: 4
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-cluster'

  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-execution-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-execution-role'

  ECSTaskRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
```

```
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-role'

  # CloudWatch Log Group
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub '/ecs/${AWS::StackName}'
      RetentionInDays: 7

  # ECS Task Definition
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Sub '${AWS::StackName}-task'
      Cpu: '256'
      Memory: '512'
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      TaskRoleArn: !GetAtt ECSTaskRole.Arn
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Image: !Ref ContainerImage
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          Essential: true
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          HealthCheck:
            Command:
              - CMD-SHELL
```

```
            - curl -f http://localhost/ || exit 1
          Interval: 30
          Timeout: 5
          Retries: 3
          StartPeriod: 60
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task'

  # ECS Service
  ECSService:
    Type: AWS::ECS::Service
    DependsOn: ALBListener
    Properties:
      ServiceName: !Sub '${AWS::StackName}-service'
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref TaskDefinition
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      PlatformVersion: LATEST
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: DISABLED
          SecurityGroups:
            - !Ref ECSSecurityGroup
          Subnets:
            - !Ref PrivateSubnet1
            - !Ref PrivateSubnet2
      LoadBalancers:
        - ContainerName: !Ref ServiceName
          ContainerPort: !Ref ContainerPort
          TargetGroupArn: !Ref ALBTargetGroup
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 50
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      EnableExecuteCommand: true  # For debugging
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-service'

  # Auto Scaling Target
```

```
  ServiceScalingTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    Properties:
      MaxCapacity: !Ref MaxCapacity
      MinCapacity: !Ref MinCapacity
      ResourceId: !Sub 'service/${ECSCluster}/${ECSService.Name}'
      RoleARN: !Sub 'arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService'
      ScalableDimension: ecs:service:DesiredCount
      ServiceNamespace: ecs

  # Auto Scaling Policy - CPU Utilization
  ServiceScalingPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    Properties:
      PolicyName: !Sub '${AWS::StackName}-cpu-scaling-policy'
      PolicyType: TargetTrackingScaling
      ScalingTargetId: !Ref ServiceScalingTarget
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ECSServiceAverageCPUUtilization
        TargetValue: 70.0
        ScaleOutCooldown: 300
        ScaleInCooldown: 300

Outputs:
  VPCId:
    Description: VPC ID
    Value: !Ref VPC
    Export:
      Name: !Sub '${AWS::StackName}-VPC-ID'

  LoadBalancerURL:
    Description: URL of the Application Load Balancer
    Value: !Sub 'http://${ApplicationLoadBalancer.DNSName}'
    Export:
      Name: !Sub '${AWS::StackName}-ALB-URL'

  ECSClusterName:
    Description: Name of the ECS Cluster
    Value: !Ref ECSCluster
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Cluster'
```

```
  ECSServiceName:
    Description: Name of the ECS Service
    Value: !GetAtt ECSService.Name
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Service'

  PrivateSubnet1:
    Description: Private Subnet 1 ID
    Value: !Ref PrivateSubnet1
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-1'

  PrivateSubnet2:
    Description: Private Subnet 2 ID
    Value: !Ref PrivateSubnet2
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-2'
```

The template used in this tutorial creates an Amazon ECS service with two tasks that run on Fargate. The tasks each run a sample Amazon ECS application. The template also creates an Application Load Balancer that distributes application traffic and an Application Auto Scaling policy that scales the application based on CPU utilization. The template also creates the networking resources necessary to deploy the application, the logging resources for container logs, and an Amazon ECS task execution IAM role. For more information about the task execution role, see Amazon ECS task execution IAM role. For more information about auto scaling, see Automatically scale your Amazon ECS service.

Step 2: Create a stack for Amazon ECS resources

After creating a file for the template, you can follow these steps to create a stack with the template by using the AWS CloudFormation console.

For information about how to create a stack using the AWS CloudFormation console, see Creating a stack on the AWS CloudFormation console in the *AWS CloudFormation User Guide* and use the following table to determine what options to specify.

Option	Value	
Prerequisite - Prepare template	Choose an existing template	
Specify template	Upload a template file	
Choose file	ecs-tutorial-template.yaml	
Stack name	ecs-tutorial-stack	
Parameters	Leave all parameter values as defaults.	
Capabilities	Choose **I acknowledge that this template may create IAM resources** to acknowledge AWS CloudFormation creating IAM resources.	

Step 3: Verify

Use the following steps to verify the creation of Amazon ECS resources using the provided template.

For information about how to view stack information and resources, see Viewing stack information from the CloudFormation console in the *AWS CloudFormation User Guide* and use the following table to determine what to verify.

Stack details field	What to look for	
Stack info	A status of `CREATE_COMPLETE` .	
Resources	A list of the created resources with links to service console. Choose links to `ECSCluster` , `ECSServic`	

Stack details field	What to look for	
	`e` , `TaskDefinition` to view more details about the created service, cluster, and task definition in the Amazon ECS console.	
Outputs	**LoadBalancerURL**. Paste the URL into a web browser to view a webpage that displays a sample Amazon ECS application.	

Step 4: Clean up resources

To clean up resources and avoid incurring further costs, follow the steps in Delete a stack from the CloudFormation console in the *AWS CloudFormation user guide*.

Creating Amazon ECS resources using AWS CLI commands for AWS CloudFormation

Another way to use Amazon ECS with AWS CloudFormation is through the AWS CLI. You can use commands to create your AWS CloudFormation stacks for Amazon ECS components like task definitions, clusters, and services and deploy them. The following tutorial shows how you can use the AWS CLI to create Amazon ECS resources using an AWS CloudFormation template.

Prerequisites

- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.

Step 1: Create a stack

To create a stack using the AWS CLI saved in a file called `ecs-tutorial-template.yaml`, run the following command.

```
cat << 'EOF' > ecs-tutorial-template.yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] ECS: load-balanced-web-application'
Parameters:
  VpcCidr:
    Type: String
    Default: '10.0.0.0/16'
    Description: CIDR block for the VPC
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest'
    Description: Container image to use in task definition

  PublicSubnet1Cidr:
    Type: String
    Default: '10.0.1.0/24'
    Description: CIDR block for public subnet 1

  PublicSubnet2Cidr:
    Type: String
    Default: '10.0.2.0/24'
    Description: CIDR block for public subnet 2

  PrivateSubnet1Cidr:
    Type: String
    Default: '10.0.3.0/24'
    Description: CIDR block for private subnet 1

  PrivateSubnet2Cidr:
    Type: String
    Default: '10.0.4.0/24'
    Description: CIDR block for private subnet 2

  ServiceName:
    Type: String
    Default: 'tutorial-app'
    Description: Name of the ECS service
```

```
  ContainerPort:
    Type: Number
    Default: 80
    Description: Port on which the container listens

  DesiredCount:
    Type: Number
    Default: 2
    Description: Desired number of tasks

  MinCapacity:
    Type: Number
    Default: 1
    Description: Minimum number of tasks for auto scaling

  MaxCapacity:
    Type: Number
    Default: 10
    Description: Maximum number of tasks for auto scaling

Resources:
  # VPC and Networking
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-vpc'

  # Internet Gateway
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-igw'

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
```

```
      VpcId: !Ref VPC

  # Public Subnets for ALB
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PublicSubnet1Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-1'

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PublicSubnet2Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-2'

  # Private Subnets for ECS Tasks
  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet1Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-1'

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet2Cidr
      Tags:
        - Key: Name
```

```
        Value: !Sub '${AWS::StackName}-private-subnet-2'

  # NAT Gateways for private subnet internet access
  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-1'

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-2'

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-1'

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-2'

  # Route Tables
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
```

```
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-routes'

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-1'

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1
```

```
  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-2'

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  # Security Groups
  ALBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-alb-sg'
      GroupDescription: Security group for Application Load Balancer
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
          Description: Allow HTTP traffic from internet
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb-sg'

  ECSSecurityGroup:
```

```
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-ecs-sg'
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: !Ref ContainerPort
          ToPort: !Ref ContainerPort
          SourceSecurityGroupId: !Ref ALBSecurityGroup
          Description: Allow traffic from ALB
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-ecs-sg'

  # Application Load Balancer
  ApplicationLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: !Sub '${AWS::StackName}-alb'
      Scheme: internet-facing
      Type: application
      Subnets:
        - !Ref PublicSubnet1
        - !Ref PublicSubnet2
      SecurityGroups:
        - !Ref ALBSecurityGroup
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb'

  ALBTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: !Sub '${AWS::StackName}-tg'
      Port: !Ref ContainerPort
      Protocol: HTTP
      VpcId: !Ref VPC
      TargetType: ip
      HealthCheckIntervalSeconds: 30
```

```
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      UnhealthyThresholdCount: 5
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-tg'

  ALBListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref ALBTargetGroup
      LoadBalancerArn: !Ref ApplicationLoadBalancer
      Port: 80
      Protocol: HTTP

  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Sub '${AWS::StackName}-cluster'
      CapacityProviders:
        - FARGATE
        - FARGATE_SPOT
      DefaultCapacityProviderStrategy:
        - CapacityProvider: FARGATE
          Weight: 1
        - CapacityProvider: FARGATE_SPOT
          Weight: 4
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-cluster'

  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-execution-role'
```

```
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-execution-role'

  ECSTaskRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-role'

  # CloudWatch Log Group
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub '/ecs/${AWS::StackName}'
      RetentionInDays: 7

  # ECS Task Definition
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Sub '${AWS::StackName}-task'
      Cpu: '256'
      Memory: '512'
      NetworkMode: awsvpc
      RequiresCompatibilities:
```

```
        - FARGATE
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      TaskRoleArn: !GetAtt ECSTaskRole.Arn
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Image: !Ref ContainerImage
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          Essential: true
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          HealthCheck:
            Command:
              - CMD-SHELL
              - curl -f http://localhost/ || exit 1
            Interval: 30
            Timeout: 5
            Retries: 3
            StartPeriod: 60
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task'

  # ECS Service
  ECSService:
    Type: AWS::ECS::Service
    DependsOn: ALBListener
    Properties:
      ServiceName: !Sub '${AWS::StackName}-service'
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref TaskDefinition
      DesiredCount: !Ref DesiredCount
      LaunchType: FARGATE
      PlatformVersion: LATEST
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: DISABLED
          SecurityGroups:
            - !Ref ECSSecurityGroup
```

```
          Subnets:
            - !Ref PrivateSubnet1
            - !Ref PrivateSubnet2
      LoadBalancers:
        - ContainerName: !Ref ServiceName
          ContainerPort: !Ref ContainerPort
          TargetGroupArn: !Ref ALBTargetGroup
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 50
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      EnableExecuteCommand: true  # For debugging
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-service'

  # Auto Scaling Target
  ServiceScalingTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    Properties:
      MaxCapacity: !Ref MaxCapacity
      MinCapacity: !Ref MinCapacity
      ResourceId: !Sub 'service/${ECSCluster}/${ECSService.Name}'
      RoleARN: !Sub 'arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService'
      ScalableDimension: ecs:service:DesiredCount
      ServiceNamespace: ecs

  # Auto Scaling Policy - CPU Utilization
  ServiceScalingPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    Properties:
      PolicyName: !Sub '${AWS::StackName}-cpu-scaling-policy'
      PolicyType: TargetTrackingScaling
      ScalingTargetId: !Ref ServiceScalingTarget
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ECSServiceAverageCPUUtilization
        TargetValue: 70.0
        ScaleOutCooldown: 300
        ScaleInCooldown: 300
```

```
Outputs:
  VPCId:
    Description: VPC ID
    Value: !Ref VPC
    Export:
      Name: !Sub '${AWS::StackName}-VPC-ID'

  LoadBalancerURL:
    Description: URL of the Application Load Balancer
    Value: !Sub 'http://${ApplicationLoadBalancer.DNSName}'
    Export:
      Name: !Sub '${AWS::StackName}-ALB-URL'

  ECSClusterName:
    Description: Name of the ECS Cluster
    Value: !Ref ECSCluster
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Cluster'

  ECSServiceName:
    Description: Name of the ECS Service
    Value: !GetAtt ECSService.Name
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Service'

  PrivateSubnet1:
    Description: Private Subnet 1 ID
    Value: !Ref PrivateSubnet1
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-1'

  PrivateSubnet2:
    Description: Private Subnet 2 ID
    Value: !Ref PrivateSubnet2
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-2'
EOF
```

The template used in this tutorial creates an Amazon ECS service with two tasks that run on Fargate. The tasks each run a sample Amazon ECS application. The template also creates an Application Load Balancer that distributes application traffic and an Application Auto Scaling policy that scales the application based on CPU utilization. The template also creates the networking resources necessary to deploy the application, the logging resources for container logs, and an Amazon ECS task execution IAM role. For more information about the task execution role, see Amazon ECS task execution IAM role. For more information about auto scaling, see Automatically scale your Amazon ECS service.

After creating a template file, use the following command to create a stack. The `--capabilities` flag is required to create an Amazon ECS task execution role as specified in the template. You can also specify the `--parameters` flag to customize the template parameters.

```
aws cloudformation create-stack \
      --stack-name ecs-tutorial-stack \
      --template-body file://ecs-tutorial-template.yaml \
      --region aws-region \
      --capabilities CAPABILITY_NAMED_IAM
```

After running the `create-stack` command, you can use `describe-stacks` to check the status of stack creation.

```
aws cloudformation describe-stacks \
      --stack-name ecs-tutorial-stack \
      --region aws-region
```

Step 2: Verify Amazon ECS resource creation

To ensure that Amazon ECS resources are created correctly, follow these steps.

1. Run the following command to list all task definitions in an AWS Region.

   ```
   aws ecs list-task-definitions
   ```

   The command returns a list of task definition Amazon Resource Name (ARN)s. The ARN of the task definition that you created using the template will be displayed in the following format.

   ```
   {
       "taskDefinitionArns": [
        .....
           "arn:aws:ecs:aws-region:111122223333:task-definition/ecs-tutorial-stack-
   task:1",
        .....
       ]
   ```

```
}
```

2. Run the following command to list all clusters in an AWS Region.

```
aws ecs list-clusters
```

   The command returns a list of cluster ARNs. The ARN of the cluster that you created using the template will be displayed in the following format.

```
{
    "clusterArns": [
        .....
        "arn:aws:ecs:aws-region:111122223333:cluster/ecs-tutorial-stack-cluster",
        .....
    ]
}
```

3. Run the following command to list all services in the cluster `ecs-tutorial-stack-cluster`.

```
aws ecs list-services \
      --cluster ecs-tutorial-stack-cluster
```

   The command returns a list of service ARNs. The ARN of the service that you created using the template will be displayed in the following format.

```
{
    "serviceArns": [
        "arn:aws:ecs:aws-region:111122223333:service/ecs-tutorial-stack-cluster/
ecs-tutorial-stack-service"
    ]
}
```

You can also obtain the DNS name of the Application Load Balancer that was created and use it to verify the creation of resources. To obtain the DNS name, run the following command:

Run the following command to retrieve outputs of the created stack.

```
aws cloudformation describe-stacks \
  --stack-name ecs-tutorial-stack \
```

```
  --region aws-region \
  --query 'Stacks[0].Outputs[?OutputKey==`LoadBalancerURL`].OutputValue' \
  --output  text
```

Output:

```
http://ecs-tutorial-stack-alb-0123456789.aws-region.elb.amazonaws.com
```

Paste the DNS name into a browser to view a webpage that displays a sample Amazon ECS application.

Step 3: Clean up

To clean up the resources you created, run the following command.

```
aws cloudformation delete-stack \
      --stack-name ecs-stack
```

The `delete-stack` command initiates deletion of the AWS CloudFormation stack that was created in this tutorial, deleting all the resources in the stack. To verify deletion, you can repeat the procedure in Step 2: Verify Amazon ECS resource creation. The list of ARNs in the outputs will no longer include a task definition called `ecs-tutorial-stack-task` or a cluster called `ecs-tutorial-stack-cluster`. The `list-services` call will fail.

AWS CloudFormation example templates for Amazon ECS

You can create Amazon ECS clusters, task definitions, and services using AWS CloudFormation. The following topics include templates that demonstrate how to create resources with different configurations. You can create these resources with these templates by using the AWS CloudFormation console or the AWS CLI.

AWS CloudFormation templates are text files in the JSON or YAML format that describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with either the JSON or YAML format, or both, you can use AWS Infrastructure Composer to get started using AWS CloudFormation templates. For more information, see Create templates visually with Infrastructure Composer in the *AWS CloudFormation User Guide*.

The following topics list example templates for Amazon ECS task definitions, clusters, and services.

Topics

- Task definitions
- Clusters
- Services
- IAM roles for Amazon ECS

Task definitions

A task definition is a blueprint for your application that describes the parameters and one or more containers that form your application. The following are example AWS CloudFormation templates for Amazon ECS task definitions. For more information about Amazon ECS task definitions, see Amazon ECS task definitions.

Fargate Linux task definition

You can use the following template to create a sample Fargate Linux task.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "ECS Task Definition with parameterized values",
  "Parameters": {
    "ContainerImage": {
      "Type": "String",
      "Default": "public.ecr.aws/docker/library/httpd:2.4",
      "Description": "The container image to use for the task"
    },
    "ContainerCpu": {
      "Type": "Number",
      "Default": 256,
      "Description": "The number of CPU units to reserve for the container",
      "AllowedValues": [256, 512, 1024, 2048, 4096]
    },
    "ContainerMemory": {
      "Type": "Number",
      "Default": 512,
      "Description": "The amount of memory (in MiB) to reserve for the container",
      "AllowedValues": [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
    },
    "TaskFamily": {
      "Type": "String",
```

```
      "Default": "task-definition-cfn",
      "Description": "The name of the task definition family"
    },
    "ContainerName": {
      "Type": "String",
      "Default": "sample-fargate-app",
      "Description": "The name of the container"
    },
    "ContainerPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the container"
    },
    "HostPort": {
      "Type": "Number",
      "Default": 80,
      "Description": "The port number on the host"
    },
    "ExecutionRoleArn": {
      "Type": "String",
      "Default": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole",
      "Description": "The ARN of the task execution role"
    },
    "LogGroup": {
      "Type": "String",
      "Default": "/ecs/fargate-task-definition",
      "Description": "The CloudWatch log group for container logs"
    },
    "NetworkMode": {
      "Type": "String",
      "Default": "awsvpc",
      "Description": "The Docker networking mode to use",
      "AllowedValues": ["awsvpc", "bridge", "host", "none"]
    },
    "OperatingSystemFamily": {
      "Type": "String",
      "Default": "LINUX",
      "Description": "The operating system for the task",
      "AllowedValues": ["LINUX", "WINDOWS_SERVER_2019_FULL",
 "WINDOWS_SERVER_2019_CORE", "WINDOWS_SERVER_2022_FULL", "WINDOWS_SERVER_2022_CORE"]
    }
  },
  "Resources": {
    "ECSTaskDefinition": {
```

```
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ContainerDefinitions": [
          {
            "Command": [
              "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && -
foreground\""
            ],
            "EntryPoint": [
              "sh",
              "-c"
            ],
            "Essential": true,
            "Image": {"Ref": "ContainerImage"},
            "LogConfiguration": {
              "LogDriver": "awslogs",
              "Options": {
                "mode": "non-blocking",
                "max-buffer-size": "25m",
                "awslogs-create-group": "true",
                "awslogs-group": {"Ref": "LogGroup"},
                "awslogs-region": {"Ref": "AWS::Region"},
                "awslogs-stream-prefix": "ecs"
              }
            },
            "Name": {"Ref": "ContainerName"},
            "PortMappings": [
              {
                "ContainerPort": {"Ref": "ContainerPort"},
                "HostPort": {"Ref": "HostPort"},
                "Protocol": "tcp"
              }
            ]
          }
        ],
        "Cpu": {"Ref": "ContainerCpu"},
        "ExecutionRoleArn": {"Ref": "ExecutionRoleArn"},
        "Family": {"Ref": "TaskFamily"},
        "Memory": {"Ref": "ContainerMemory"},
        "NetworkMode": {"Ref": "NetworkMode"},
```

```
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "RuntimePlatform": {
          "OperatingSystemFamily": {"Ref": "OperatingSystemFamily"}
        }
      }
    }
  },
  "Outputs": {
    "TaskDefinitionArn": {
      "Description": "The ARN of the created task definition",
      "Value": {"Ref": "ECSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: 'ECS Task Definition to deploy a sample app'
Parameters:
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/docker/library/httpd:2.4'
    Description: The container image to use for the task
  ContainerCpu:
    Type: Number
    Default: 256
    Description: The number of CPU units to reserve for the container
    AllowedValues: [256, 512, 1024, 2048, 4096]
  ContainerMemory:
    Type: Number
    Default: 512
    Description: The amount of memory (in MiB) to reserve for the container
    AllowedValues: [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
  TaskFamily:
    Type: String
    Default: 'task-definition-cfn'
    Description: The name of the task definition family
  ContainerName:
    Type: String
    Default: 'sample-fargate-app'
```

```
      Description: The name of the container
    ContainerPort:
      Type: Number
      Default: 80
      Description: The port number on the container
    HostPort:
      Type: Number
      Default: 80
      Description: The port number on the host
    ExecutionRoleArn:
      Type: String
      Default: 'arn:aws:iam::111122223333:role/ecsTaskExecutionRole'
      Description: The ARN of the task execution role
    LogGroup:
      Type: String
      Default: '/ecs/fargate-task-definition'
      Description: The CloudWatch log group for container logs
    NetworkMode:
      Type: String
      Default: 'awsvpc'
      Description: The Docker networking mode to use
      AllowedValues: ['awsvpc', 'bridge', 'host', 'none']
    OperatingSystemFamily:
      Type: String
      Default: 'LINUX'
      Description: The operating system for the task
      AllowedValues: ['LINUX', 'WINDOWS_SERVER_2019_FULL', 'WINDOWS_SERVER_2019_CORE',
   'WINDOWS_SERVER_2022_FULL', 'WINDOWS_SERVER_2022_CORE']
Resources:
    ECSTaskDefinition:
      Type: 'AWS::ECS::TaskDefinition'
      Properties:
        ContainerDefinitions:
          - Command:
              - >-
                /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
                App</title> <style>body {margin-top: 40px; background-color:
                #333;} </style> </head><body> <div
                style=color:white;text-align:center> <h1>Amazon ECS Sample
                App</h1> <h2>Congratulations!</h2> <p>Your application is now
                running on a container in Amazon ECS.</p> </div></body></html>' >
                /usr/local/apache2/htdocs/index.html && httpd-foreground"
            EntryPoint:
              - sh
```

```
            - '-c'
          Essential: true
          Image: !Ref ContainerImage
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-create-group: 'true'
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
          Name: !Ref ContainerName
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              HostPort: !Ref HostPort
              Protocol: tcp
      Cpu: !Ref ContainerCpu
      ExecutionRoleArn: !Ref ExecutionRoleArn
      Family: !Ref TaskFamily
      Memory: !Ref ContainerMemory
      NetworkMode: !Ref NetworkMode
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: !Ref OperatingSystemFamily
Outputs:
  TaskDefinitionArn:
    Description: The ARN of the created task definition
    Value: !Ref ECSTaskDefinition
```

Amazon EFS task definition

You can use the following template to create a task that uses an Amazon EFS file system that you created. For more information about using Amazon EFS volumes with Amazon ECS, see Use Amazon EFS volumes with Amazon ECS.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
```

```
 "Description": "Create a task definition for a web server with parameterized
values.",
 "Parameters": {
   "ExecutionRoleArn": {
     "Type": "String",
     "Default": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
     "Description": "The ARN of the task execution role"
   },
   "NetworkMode": {
     "Type": "String",
     "Default": "awsvpc",
     "Description": "The Docker networking mode to use",
     "AllowedValues": ["awsvpc", "bridge", "host", "none"]
   },
   "TaskFamily": {
     "Type": "String",
     "Default": "my-ecs-task",
     "Description": "The name of the task definition family"
   },
   "ContainerCpu": {
     "Type": "String",
     "Default": "256",
     "Description": "The number of CPU units to reserve for the container",
     "AllowedValues": ["256", "512", "1024", "2048", "4096"]
   },
   "ContainerMemory": {
     "Type": "String",
     "Default": "512",
     "Description": "The amount of memory (in MiB) to reserve for the container",
     "AllowedValues": ["512", "1024", "2048", "3072", "4096", "5120", "6144",
"7168", "8192"]
   },
   "ContainerName": {
     "Type": "String",
     "Default": "nginx",
     "Description": "The name of the container"
   },
   "ContainerImage": {
     "Type": "String",
     "Default": "public.ecr.aws/nginx/nginx:latest",
     "Description": "The container image to use for the task"
   },
   "ContainerPort": {
     "Type": "Number",
```

```
      "Default": 80,
      "Description": "The port number on the container"
    },
    "InitProcessEnabled": {
      "Type": "String",
      "Default": "true",
      "Description": "Whether to enable the init process inside the container",
      "AllowedValues": ["true", "false"]
    },
    "EfsVolumeName": {
      "Type": "String",
      "Default": "efs-volume",
      "Description": "The name of the EFS volume"
    },
    "EfsContainerPath": {
      "Type": "String",
      "Default": "/usr/share/nginx/html",
      "Description": "The path in the container where the EFS volume will be
 mounted"
    },
    "LogGroup": {
      "Type": "String",
      "Default": "LogGroup",
      "Description": "The CloudWatch log group for container logs"
    },
    "LogStreamPrefix": {
      "Type": "String",
      "Default": "efs-task",
      "Description": "The prefix for the log stream"
    },
    "EfsFilesystemId": {
      "Type": "String",
      "Default": "fs-1234567890abcdef0",
      "Description": "The ID of the EFS filesystem"
    },
    "EfsRootDirectory": {
      "Type": "String",
      "Default": "/",
      "Description": "The root directory in the EFS filesystem"
    },
    "EfsTransitEncryption": {
      "Type": "String",
      "Default": "ENABLED",
      "Description": "Whether to enable transit encryption for EFS",
```

```
      "AllowedValues": ["ENABLED", "DISABLED"]
    }
  },
  "Resources": {
    "ECSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Ref": "ExecutionRoleArn"},
        "NetworkMode": {"Ref": "NetworkMode"},
        "RequiresCompatibilities": ["FARGATE"],
        "Family": {"Ref": "TaskFamily"},
        "Cpu": {"Ref": "ContainerCpu"},
        "Memory": {"Ref": "ContainerMemory"},
        "ContainerDefinitions": [
          {
            "Name": {"Ref": "ContainerName"},
            "Image": {"Ref": "ContainerImage"},
            "Essential": true,
            "PortMappings": [
              {
                "ContainerPort": {"Ref": "ContainerPort"},
                "Protocol": "tcp"
              }
            ],
            "LinuxParameters": {
              "InitProcessEnabled": {"Ref": "InitProcessEnabled"}
            },
            "MountPoints": [
              {
                "SourceVolume": {"Ref": "EfsVolumeName"},
                "ContainerPath": {"Ref": "EfsContainerPath"}
              }
            ],
            "LogConfiguration": {
              "LogDriver": "awslogs",
              "Options": {
                "mode": "non-blocking",
                "max-buffer-size": "25m",
                "awslogs-group": {"Ref": "LogGroup"},
                "awslogs-region": {"Ref": "AWS::Region"},
                "awslogs-create-group": "true",
                "awslogs-stream-prefix": {"Ref": "LogStreamPrefix"}
              }
            }
```

```
          }
        ],
        "Volumes": [
          {
            "Name": {"Ref": "EfsVolumeName"},
            "EFSVolumeConfiguration": {
              "FilesystemId": {"Ref": "EfsFilesystemId"},
              "RootDirectory": {"Ref": "EfsRootDirectory"},
              "TransitEncryption": {"Ref": "EfsTransitEncryption"}
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "TaskDefinitionArn": {
      "Description": "The ARN of the created task definition",
      "Value": {"Ref": "ECSTaskDefinition"}
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: Create a task definition for a web server with parameterized values.
Parameters:
  ExecutionRoleArn:
    Type: String
    Default: arn:aws:iam::123456789012:role/ecsTaskExecutionRole
    Description: The ARN of the task execution role
  NetworkMode:
    Type: String
    Default: awsvpc
    Description: The Docker networking mode to use
    AllowedValues: [awsvpc, bridge, host, none]
  TaskFamily:
    Type: String
    Default: my-ecs-task
    Description: The name of the task definition family
  ContainerCpu:
    Type: String
```

```
    Default: "256"
    Description: The number of CPU units to reserve for the container
    AllowedValues: ["256", "512", "1024", "2048", "4096"]
  ContainerMemory:
    Type: String
    Default: "512"
    Description: The amount of memory (in MiB) to reserve for the container
    AllowedValues: ["512", "1024", "2048", "3072", "4096", "5120", "6144", "7168",
 "8192"]
  ContainerName:
    Type: String
    Default: nginx
    Description: The name of the container
  ContainerImage:
    Type: String
    Default: public.ecr.aws/nginx/nginx:latest
    Description: The container image to use for the task
  ContainerPort:
    Type: Number
    Default: 80
    Description: The port number on the container
  InitProcessEnabled:
    Type: String
    Default: "true"
    Description: Whether to enable the init process inside the container
    AllowedValues: ["true", "false"]
  EfsVolumeName:
    Type: String
    Default: efs-volume
    Description: The name of the EFS volume
  EfsContainerPath:
    Type: String
    Default: /usr/share/nginx/html
    Description: The path in the container where the EFS volume will be mounted
  LogGroup:
    Type: String
    Default: LogGroup
    Description: The CloudWatch log group for container logs
  LogStreamPrefix:
    Type: String
    Default: efs-task
    Description: The prefix for the log stream
  EfsFilesystemId:
    Type: String
```

```
    Default: fs-1234567890abcdef0
    Description: The ID of the EFS filesystem
  EfsRootDirectory:
    Type: String
    Default: /
    Description: The root directory in the EFS filesystem
  EfsTransitEncryption:
    Type: String
    Default: ENABLED
    Description: Whether to enable transit encryption for EFS
    AllowedValues: [ENABLED, DISABLED]
Resources:
  ECSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ExecutionRoleArn: !Ref ExecutionRoleArn
      NetworkMode: !Ref NetworkMode
      RequiresCompatibilities:
        - FARGATE
      Family: !Ref TaskFamily
      Cpu: !Ref ContainerCpu
      Memory: !Ref ContainerMemory
      ContainerDefinitions:
        - Name: !Ref ContainerName
          Image: !Ref ContainerImage
          Essential: true
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              Protocol: tcp
          LinuxParameters:
            InitProcessEnabled: !Ref InitProcessEnabled
          MountPoints:
            - SourceVolume: !Ref EfsVolumeName
              ContainerPath: !Ref EfsContainerPath
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-group: !Ref LogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-create-group: "true"
              awslogs-stream-prefix: !Ref LogStreamPrefix
      Volumes:
```

```
        - Name: !Ref EfsVolumeName
          EFSVolumeConfiguration:
            FilesystemId: !Ref EfsFilesystemId
            RootDirectory: !Ref EfsRootDirectory
            TransitEncryption: !Ref EfsTransitEncryption
Outputs:
  TaskDefinitionArn:
    Description: The ARN of the created task definition
    Value: !Ref ECSTaskDefinition
```

Clusters

An Amazon ECS cluster is a logical grouping of tasks or services. You can use the following templates to create clusters with different configurations. For more information about Amazon ECS clusters, see Amazon ECS clusters.

Create an empty cluster with default settings

You can use the following template to create an empty cluster with default settings.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": "MyEmptyCluster"
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  ECSCluster:
    Type: 'AWS::ECS::Cluster'
    Properties:
```

```
        ClusterName: MyEmptyCluster
```

Create an empty cluster with managed storage encryption and enhanced Container Insights

You can use the following template to create a cluster with cluster-level managed storage and enhanced Container Insights enabled. Cluster-level encryption applies to Amazon ECS managed data volumes such as Amazon EBS volumes. For more information about Amazon EBS encryption, see Encrypt data stored in Amazon EBS volumes attached to Amazon ECS tasks. For more information about using Container Insights with enchanced observability, see Monitor Amazon ECS containers using Container Insights with enhanced observability.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
   "Resources": {
        "Cluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": "EncryptedEnhancedCluster",
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enhanced"
                    }
                ],
                "Configuration": {
                    "ManagedStorageConfiguration": {
                        "KmsKeyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                    }
                }
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
```

```
  Cluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: EncryptedEnhancedCluster
      ClusterSettings:
        - Name: containerInsights
          Value: enhanced
      Configuration:
        ManagedStorageConfiguration:
          KmsKeyId: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

Create a cluster with the AL2023 Amazon ECS-Optimized-AMI

You can use the following template to create a cluster that uses a capacity provider that launches AL2023 instances on Amazon EC2.

> **Important**
>
> For the latest AMI IDs, see Amazon ECS-optimized AMI in the *Amazon Elastic Container Service Developer Guide*.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "EC2 ECS cluster that starts out empty, with no EC2 instances
 yet. An ECS capacity provider automatically launches more EC2 instances as required
 on the fly when you request ECS to launch services or standalone tasks.",
    "Parameters": {
        "InstanceType": {
            "Type": "String",
            "Description": "EC2 instance type",
            "Default": "t2.medium",
            "AllowedValues": [
                "t1.micro",
                "t2.2xlarge",
                "t2.large",
                "t2.medium",
                "t2.micro",
                "t2.nano",
```

```
                "t2.small",
                "t2.xlarge",
                "t3.2xlarge",
                "t3.large",
                "t3.medium",
                "t3.micro",
                "t3.nano",
                "t3.small",
                "t3.xlarge"
            ]
        },
        "DesiredCapacity": {
            "Type": "Number",
            "Default": "0",
            "Description": "Number of EC2 instances to launch in your ECS cluster."
        },
        "MaxSize": {
            "Type": "Number",
            "Default": "100",
            "Description": "Maximum number of EC2 instances that can be launched in
 your ECS cluster."
        },
        "ECSAMI": {
            "Description": "The Amazon Machine Image ID used for the cluster",
            "Type": "AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>",
            "Default": "/aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended/image_id"
        },
        "VpcId": {
            "Type": "AWS::EC2::VPC::Id",
            "Description": "VPC ID where the ECS cluster is launched",
            "Default": "vpc-1234567890abcdef0"
        },
        "SubnetIds": {
            "Type": "List<AWS::EC2::Subnet::Id>",
            "Description": "List of subnet IDs where the EC2 instances will be
 launched",
            "Default": "subnet-021345abcdef67890"
        }
    },
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
```

```
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enabled"
                    }
                ]
            }
        },
        "ECSAutoScalingGroup": {
            "Type": "AWS::AutoScaling::AutoScalingGroup",
            "DependsOn": [
                "ECSCluster",
                "EC2Role"
            ],
            "Properties": {
                "VPCZoneIdentifier": {
                    "Ref": "SubnetIds"
                },
                "LaunchTemplate": {
                    "LaunchTemplateId": {
                        "Ref": "ContainerInstances"
                    },
                    "Version": {
                        "Fn::GetAtt": [
                            "ContainerInstances",
                            "LatestVersionNumber"
                        ]
                    }
                },
                "MinSize": 0,
                "MaxSize": {
                    "Ref": "MaxSize"
                },
                "DesiredCapacity": {
                    "Ref": "DesiredCapacity"
                },
                "NewInstancesProtectedFromScaleIn": true
            },
            "UpdatePolicy": {
                "AutoScalingReplacingUpdate": {
                    "WillReplace": "true"
                }
            }
        },
```

```
        "ContainerInstances": {
            "Type": "AWS::EC2::LaunchTemplate",
            "Properties": {
                "LaunchTemplateName": "asg-launch-template-2",
                "LaunchTemplateData": {
                    "ImageId": {
                        "Ref": "ECSAMI"
                    },
                    "InstanceType": {
                        "Ref": "InstanceType"
                    },
                    "IamInstanceProfile": {
                        "Name": {
                            "Ref": "EC2InstanceProfile"
                        }
                    },
                    "SecurityGroupIds": [
                        {
                            "Ref": "ContainerHostSecurityGroup"
                        }
                    ],
                    "UserData": {
                        "Fn::Base64": {
                            "Fn::Sub": "#!/bin/bash -xe\n echo ECS_CLUSTER=
${ECSCluster} >> /etc/ecs/ecs.config\n yum install -y aws-cfn-bootstrap\n /opt/aws/
bin/cfn-init -v --stack ${AWS::StackId} --resource ContainerInstances --configsets
 full_install --region ${AWS::Region} &\n"
                        }
                    },
                    "MetadataOptions": {
                        "HttpEndpoint": "enabled",
                        "HttpTokens": "required"
                    }
                }
            }
        },
        "EC2InstanceProfile": {
            "Type": "AWS::IAM::InstanceProfile",
            "Properties": {
                "Path": "/",
                "Roles": [
                    {
                        "Ref": "EC2Role"
                    }
```

```
                ]
            }
        },
        "CapacityProvider": {
            "Type": "AWS::ECS::CapacityProvider",
            "Properties": {
                "AutoScalingGroupProvider": {
                    "AutoScalingGroupArn": {
                        "Ref": "ECSAutoScalingGroup"
                    },
                    "ManagedScaling": {
                        "InstanceWarmupPeriod": 60,
                        "MinimumScalingStepSize": 1,
                        "MaximumScalingStepSize": 100,
                        "Status": "ENABLED",
                        "TargetCapacity": 100
                    },
                    "ManagedTerminationProtection": "ENABLED"
                }
            }
        },
        "CapacityProviderAssociation": {
            "Type": "AWS::ECS::ClusterCapacityProviderAssociations",
            "Properties": {
                "CapacityProviders": [
                    {
                        "Ref": "CapacityProvider"
                    }
                ],
                "Cluster": {
                    "Ref": "ECSCluster"
                },
                "DefaultCapacityProviderStrategy": [
                    {
                        "Base": 0,
                        "CapacityProvider": {
                            "Ref": "CapacityProvider"
                        },
                        "Weight": 1
                    }
                ]
            }
        },
        "ContainerHostSecurityGroup": {
```

```
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "Access to the EC2 hosts that run containers",
                "VpcId": {
                    "Ref": "VpcId"
                }
            }
        },
        "EC2Role": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "ec2.amazonaws.com"
                                ]
                            },
                            "Action": [
                                "sts:AssumeRole"
                            ]
                        }
                    ]
                },
                "Path": "/",
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role",
                    "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
                ]
            }
        },
        "ECSTaskExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "ecs-tasks.amazonaws.com"
```

```
                                ]
                            },
                            "Action": [
                                "sts:AssumeRole"
                            ],
                            "Condition": {
                                "ArnLike": {
                                    "aws:SourceArn": {
                                        "Fn::Sub": "arn:${AWS::Partition}:ecs:
${AWS::Region}:${AWS::AccountId}:*"
                                    }
                                },
                                "StringEquals": {
                                    "aws:SourceAccount": {
                                        "Fn::Sub": "${AWS::AccountId}"
                                    }
                                }
                            }
                        }
                    ]
                },
                "Path": "/",
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy"
                ]
            }
        }
    },
    "Outputs": {
        "ClusterName": {
            "Description": "The ECS cluster into which to launch resources",
            "Value": "ECSCluster"
        },
        "ECSTaskExecutionRole": {
            "Description": "The role used to start up a task",
            "Value": "ECSTaskExecutionRole"
        },
        "CapacityProvider": {
            "Description": "The cluster capacity provider that the service should
 use to request capacity when it wants to start up a task",
            "Value": "CapacityProvider"
        }
    }
```

```
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: EC2 ECS cluster that starts out empty, with no EC2 instances yet. An
 ECS capacity provider automatically launches more EC2 instances as required on the
 fly when you request ECS to launch services or standalone tasks.
Parameters:
  InstanceType:
    Type: String
    Description: EC2 instance type
    Default: t2.medium
    AllowedValues:
      - t1.micro
      - t2.2xlarge
      - t2.large
      - t2.medium
      - t2.micro
      - t2.nano
      - t2.small
      - t2.xlarge
      - t3.2xlarge
      - t3.large
      - t3.medium
      - t3.micro
      - t3.nano
      - t3.small
      - t3.xlarge
  DesiredCapacity:
    Type: Number
    Default: '0'
    Description: Number of EC2 instances to launch in your ECS cluster.
  MaxSize:
    Type: Number
    Default: '100'
    Description: Maximum number of EC2 instances that can be launched in your ECS
 cluster.
  ECSAMI:
    Description: The Amazon Machine Image ID used for the cluster
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ecs/optimized-ami/amazon-linux-2023/recommended/image_id
  VpcId:
```

```
    Type: AWS::EC2::VPC::Id
    Description: VPC ID where the ECS cluster is launched
    Default: vpc-1234567890abcdef0
  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>
    Description: List of subnet IDs where the EC2 instances will be launched
    Default: subnet-021345abcdef67890
Resources:
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
  ECSAutoScalingGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    DependsOn:
      - ECSCluster
      - EC2Role
    Properties:
      VPCZoneIdentifier: !Ref SubnetIds
      LaunchTemplate:
        LaunchTemplateId: !Ref ContainerInstances
        Version: !GetAtt ContainerInstances.LatestVersionNumber
      MinSize: 0
      MaxSize: !Ref MaxSize
      DesiredCapacity: !Ref DesiredCapacity
      NewInstancesProtectedFromScaleIn: true
    UpdatePolicy:
      AutoScalingReplacingUpdate:
        WillReplace: 'true'
  ContainerInstances:
    Type: AWS::EC2::LaunchTemplate
    Properties:
      LaunchTemplateName: asg-launch-template-2
      LaunchTemplateData:
        ImageId: !Ref ECSAMI
        InstanceType: !Ref InstanceType
        IamInstanceProfile:
          Name: !Ref EC2InstanceProfile
        SecurityGroupIds:
          - !Ref ContainerHostSecurityGroup
        UserData: !Base64
          Fn::Sub: |
```

```
            #!/bin/bash -xe
            echo ECS_CLUSTER=${ECSCluster} >> /etc/ecs/ecs.config
            yum install -y aws-cfn-bootstrap
            /opt/aws/bin/cfn-init -v --stack ${AWS::StackId} --resource
 ContainerInstances --configsets full_install --region ${AWS::Region} &
        MetadataOptions:
          HttpEndpoint: enabled
          HttpTokens: required
  EC2InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: /
      Roles:
        - !Ref EC2Role
  CapacityProvider:
    Type: AWS::ECS::CapacityProvider
    Properties:
      AutoScalingGroupProvider:
        AutoScalingGroupArn: !Ref ECSAutoScalingGroup
        ManagedScaling:
          InstanceWarmupPeriod: 60
          MinimumScalingStepSize: 1
          MaximumScalingStepSize: 100
          Status: ENABLED
          TargetCapacity: 100
        ManagedTerminationProtection: ENABLED
  CapacityProviderAssociation:
    Type: AWS::ECS::ClusterCapacityProviderAssociations
    Properties:
      CapacityProviders:
        - !Ref CapacityProvider
      Cluster: !Ref ECSCluster
      DefaultCapacityProviderStrategy:
        - Base: 0
          CapacityProvider: !Ref CapacityProvider
          Weight: 1
  ContainerHostSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the EC2 hosts that run containers
      VpcId: !Ref VpcId
  EC2Role:
    Type: AWS::IAM::Role
    Properties:
```

```
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
        - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ecs-tasks.amazonaws.com
            Action:
              - sts:AssumeRole
            Condition:
              ArnLike:
                aws:SourceArn: !Sub arn:${AWS::Partition}:ecs:${AWS::Region}:
${AWS::AccountId}:*
              StringEquals:
                aws:SourceAccount: !Sub ${AWS::AccountId}
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
Outputs:
  ClusterName:
    Description: The ECS cluster into which to launch resources
    Value: ECSCluster
  ECSTaskExecutionRole:
    Description: The role used to start up a task
    Value: ECSTaskExecutionRole
  CapacityProvider:
    Description: The cluster capacity provider that the service should use to
 request capacity when it wants to start up a task
    Value: CapacityProvider
```

Services

You can use an Amazon ECS service to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. If one of your tasks fails or stops, the Amazon ECS service scheduler launches another instance of your task definition to replace it. This helps maintain your desired number of tasks in the service. The following templates can be used to deploy services. For more information about Amazon ECS services, see Amazon ECS services.

Deploy a load balanced web application

The following template creates an Amazon ECS service with two tasks that run on Fargate. The tasks each have an NGINX container. The template also creates an Application Load Balancer that distributes application traffic and an Application Auto Scaling policy that scales the application based on CPU utilization. The template also creates the networking resources necessary to deploy the application, the logging resources for container logs, and an Amazon ECS task execution IAM role. For more information about the task execution role, see Amazon ECS task execution IAM role. For more information about auto scaling, see Automatically scale your Amazon ECS service.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "[AWSDocs] ECS: load-balanced-web-application",
    "Parameters": {
        "VpcCidr": {
            "Type": "String",
            "Default": "10.0.0.0/16",
            "Description": "CIDR block for the VPC"
        },
        "ContainerImage": {
            "Type": "String",
            "Default": "public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest",
            "Description": "Container image to use in task definition"
        },
        "PublicSubnet1Cidr": {
            "Type": "String",
            "Default": "10.0.1.0/24",
            "Description": "CIDR block for public subnet 1"
        },
        "PublicSubnet2Cidr": {
            "Type": "String",
```

```
                "Default": "10.0.2.0/24",
                "Description": "CIDR block for public subnet 2"
            },
            "PrivateSubnet1Cidr": {
                "Type": "String",
                "Default": "10.0.3.0/24",
                "Description": "CIDR block for private subnet 1"
            },
            "PrivateSubnet2Cidr": {
                "Type": "String",
                "Default": "10.0.4.0/24",
                "Description": "CIDR block for private subnet 2"
            },
            "ServiceName": {
                "Type": "String",
                "Default": "tutorial-app",
                "Description": "Name of the ECS service"
            },
            "ContainerPort": {
                "Type": "Number",
                "Default": 80,
                "Description": "Port on which the container listens"
            },
            "DesiredCount": {
                "Type": "Number",
                "Default": 2,
                "Description": "Desired number of tasks"
            },
            "MinCapacity": {
                "Type": "Number",
                "Default": 1,
                "Description": "Minimum number of tasks for auto scaling"
            },
            "MaxCapacity": {
                "Type": "Number",
                "Default": 10,
                "Description": "Maximum number of tasks for auto scaling"
            }
        },
        "Resources": {
            "VPC": {
                "Type": "AWS::EC2::VPC",
                "Properties": {
                    "CidrBlock": {
```

```
                    "Ref": "VpcCidr"
                },
                "EnableDnsHostnames": true,
                "EnableDnsSupport": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-vpc"
                        }
                    }
                ]
            }
        },
        "InternetGateway": {
            "Type": "AWS::EC2::InternetGateway",
            "Properties": {
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-igw"
                        }
                    }
                ]
            }
        },
        "InternetGatewayAttachment": {
            "Type": "AWS::EC2::VPCGatewayAttachment",
            "Properties": {
                "InternetGatewayId": {
                    "Ref": "InternetGateway"
                },
                "VpcId": {
                    "Ref": "VPC"
                }
            }
        },
        "PublicSubnet1": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
```

```
                "AvailabilityZone": {
                    "Fn::Select": [
                        0,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PublicSubnet1Cidr"
                },
                "MapPublicIpOnLaunch": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-public-subnet-1"
                        }
                    }
                ]
            }
        },
        "PublicSubnet2": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "AvailabilityZone": {
                    "Fn::Select": [
                        1,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PublicSubnet2Cidr"
                },
                "MapPublicIpOnLaunch": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
```

```
                            "Fn::Sub": "${AWS::StackName}-public-subnet-2"
                        }
                    }
                ]
            }
        },
        "PrivateSubnet1": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "AvailabilityZone": {
                    "Fn::Select": [
                        0,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PrivateSubnet1Cidr"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-private-subnet-1"
                        }
                    }
                ]
            }
        },
        "PrivateSubnet2": {
            "Type": "AWS::EC2::Subnet",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "AvailabilityZone": {
                    "Fn::Select": [
                        1,
                        {
                            "Fn::GetAZs": ""
```

```
                    }
                ]
            },
            "CidrBlock": {
                "Ref": "PrivateSubnet2Cidr"
            },
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-private-subnet-2"
                    }
                }
            ]
        }
    },
    "NatGateway1EIP": {
        "Type": "AWS::EC2::EIP",
        "DependsOn": "InternetGatewayAttachment",
        "Properties": {
            "Domain": "vpc",
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-nat-eip-1"
                    }
                }
            ]
        }
    },
    "NatGateway2EIP": {
        "Type": "AWS::EC2::EIP",
        "DependsOn": "InternetGatewayAttachment",
        "Properties": {
            "Domain": "vpc",
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-nat-eip-2"
                    }
                }
            ]
```

```
            }
        },
        "NatGateway1": {
            "Type": "AWS::EC2::NatGateway",
            "Properties": {
                "AllocationId": {
                    "Fn::GetAtt": [
                        "NatGateway1EIP",
                        "AllocationId"
                    ]
                },
                "SubnetId": {
                    "Ref": "PublicSubnet1"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-nat-1"
                        }
                    }
                ]
            }
        },
        "NatGateway2": {
            "Type": "AWS::EC2::NatGateway",
            "Properties": {
                "AllocationId": {
                    "Fn::GetAtt": [
                        "NatGateway2EIP",
                        "AllocationId"
                    ]
                },
                "SubnetId": {
                    "Ref": "PublicSubnet2"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-nat-2"
                        }
                    }
                ]
```

```
            }
        },
        "PublicRouteTable": {
            "Type": "AWS::EC2::RouteTable",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-public-routes"
                        }
                    }
                ]
            }
        },
        "DefaultPublicRoute": {
            "Type": "AWS::EC2::Route",
            "DependsOn": "InternetGatewayAttachment",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "DestinationCidrBlock": "0.0.0.0/0",
                "GatewayId": {
                    "Ref": "InternetGateway"
                }
            }
        },
        "PublicSubnet1RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "SubnetId": {
                    "Ref": "PublicSubnet1"
                }
            }
        },
        "PublicSubnet2RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
```

```
            "Properties": {
                "RouteTableId": {
                    "Ref": "PublicRouteTable"
                },
                "SubnetId": {
                    "Ref": "PublicSubnet2"
                }
            }
        },
        "PrivateRouteTable1": {
            "Type": "AWS::EC2::RouteTable",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-private-routes-1"
                        }
                    }
                ]
            }
        },
        "DefaultPrivateRoute1": {
            "Type": "AWS::EC2::Route",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable1"
                },
                "DestinationCidrBlock": "0.0.0.0/0",
                "NatGatewayId": {
                    "Ref": "NatGateway1"
                }
            }
        },
        "PrivateSubnet1RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable1"
                },
                "SubnetId": {
```

```
                    "Ref": "PrivateSubnet1"
                }
            }
        },
        "PrivateRouteTable2": {
            "Type": "AWS::EC2::RouteTable",
            "Properties": {
                "VpcId": {
                    "Ref": "VPC"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-private-routes-2"
                        }
                    }
                ]
            }
        },
        "DefaultPrivateRoute2": {
            "Type": "AWS::EC2::Route",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable2"
                },
                "DestinationCidrBlock": "0.0.0.0/0",
                "NatGatewayId": {
                    "Ref": "NatGateway2"
                }
            }
        },
        "PrivateSubnet2RouteTableAssociation": {
            "Type": "AWS::EC2::SubnetRouteTableAssociation",
            "Properties": {
                "RouteTableId": {
                    "Ref": "PrivateRouteTable2"
                },
                "SubnetId": {
                    "Ref": "PrivateSubnet2"
                }
            }
        },
        "ALBSecurityGroup": {
```

```
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupName": {
                    "Fn::Sub": "${AWS::StackName}-alb-sg"
                },
                "GroupDescription": "Security group for Application Load Balancer",
                "VpcId": {
                    "Ref": "VPC"
                },
                "SecurityGroupIngress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": 80,
                        "ToPort": 80,
                        "CidrIp": "0.0.0.0/0",
                        "Description": "Allow HTTP traffic from internet"
                    }
                ],
                "SecurityGroupEgress": [
                    {
                        "IpProtocol": -1,
                        "CidrIp": "0.0.0.0/0",
                        "Description": "Allow all outbound traffic"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-alb-sg"
                        }
                    }
                ]
            }
        },
        "ECSSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupName": {
                    "Fn::Sub": "${AWS::StackName}-ecs-sg"
                },
                "GroupDescription": "Security group for ECS tasks",
                "VpcId": {
                    "Ref": "VPC"
```

```
                },
                "SecurityGroupIngress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": {
                            "Ref": "ContainerPort"
                        },
                        "ToPort": {
                            "Ref": "ContainerPort"
                        },
                        "SourceSecurityGroupId": {
                            "Ref": "ALBSecurityGroup"
                        },
                        "Description": "Allow traffic from ALB"
                    }
                ],
                "SecurityGroupEgress": [
                    {
                        "IpProtocol": -1,
                        "CidrIp": "0.0.0.0/0",
                        "Description": "Allow all outbound traffic"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-ecs-sg"
                        }
                    }
                ]
            }
        },
        "ApplicationLoadBalancer": {
            "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
            "Properties": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-alb"
                },
                "Scheme": "internet-facing",
                "Type": "application",
                "Subnets": [
                    {
                        "Ref": "PublicSubnet1"
```

```
                    },
                    {
                        "Ref": "PublicSubnet2"
                    }
                ],
                "SecurityGroups": [
                    {
                        "Ref": "ALBSecurityGroup"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-alb"
                        }
                    }
                ]
            }
        },
        "ALBTargetGroup": {
            "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
            "Properties": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-tg"
                },
                "Port": {
                    "Ref": "ContainerPort"
                },
                "Protocol": "HTTP",
                "VpcId": {
                    "Ref": "VPC"
                },
                "TargetType": "ip",
                "HealthCheckIntervalSeconds": 30,
                "HealthCheckPath": "/",
                "HealthCheckProtocol": "HTTP",
                "HealthCheckTimeoutSeconds": 5,
                "HealthyThresholdCount": 2,
                "UnhealthyThresholdCount": 5,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
```

```
                            "Fn::Sub": "${AWS::StackName}-tg"
                        }
                    }
                ]
            }
        },
        "ALBListener": {
            "Type": "AWS::ElasticLoadBalancingV2::Listener",
            "Properties": {
                "DefaultActions": [
                    {
                        "Type": "forward",
                        "TargetGroupArn": {
                            "Ref": "ALBTargetGroup"
                        }
                    }
                ],
                "LoadBalancerArn": {
                    "Ref": "ApplicationLoadBalancer"
                },
                "Port": 80,
                "Protocol": "HTTP"
            }
        },
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": {
                    "Fn::Sub": "${AWS::StackName}-cluster"
                },
                "CapacityProviders": [
                    "FARGATE",
                    "FARGATE_SPOT"
                ],
                "DefaultCapacityProviderStrategy": [
                    {
                        "CapacityProvider": "FARGATE",
                        "Weight": 1
                    },
                    {
                        "CapacityProvider": "FARGATE_SPOT",
                        "Weight": 4
                    }
                ],
```

```
                "ClusterSettings": [
                    {
                        "Name": "containerInsights",
                        "Value": "enabled"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-cluster"
                        }
                    }
                ]
            }
        },
        "ECSTaskExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "RoleName": {
                    "Fn::Sub": "${AWS::StackName}-task-execution-role"
                },
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "ecs-tasks.amazonaws.com"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "ManagedPolicyArns": [
                    "arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy"
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-task-execution-role"
                        }
```

```
                }
            ]
        }
    },
    "ECSTaskRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "RoleName": {
                "Fn::Sub": "${AWS::StackName}-task-role"
            },
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "ecs-tasks.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Tags": [
                {
                    "Key": "Name",
                    "Value": {
                        "Fn::Sub": "${AWS::StackName}-task-role"
                    }
                }
            ]
        }
    },
    "LogGroup": {
        "Type": "AWS::Logs::LogGroup",
        "Properties": {
            "LogGroupName": {
                "Fn::Sub": "/ecs/${AWS::StackName}"
            },
            "RetentionInDays": 7
        }
    },
    "TaskDefinition": {
        "Type": "AWS::ECS::TaskDefinition",
        "Properties": {
```

```
                "Family": {
                    "Fn::Sub": "${AWS::StackName}-task"
                },
                "Cpu": "256",
                "Memory": "512",
                "NetworkMode": "awsvpc",
                "RequiresCompatibilities": [
                    "FARGATE"
                ],
                "ExecutionRoleArn": {
                    "Fn::GetAtt": [
                        "ECSTaskExecutionRole",
                        "Arn"
                    ]
                },
                "TaskRoleArn": {
                    "Fn::GetAtt": [
                        "ECSTaskRole",
                        "Arn"
                    ]
                },
                "ContainerDefinitions": [
                    {
                        "Name": {
                            "Ref": "ServiceName"
                        },
                        "Image": {
                            "Ref": "ContainerImage"
                        },
                        "PortMappings": [
                            {
                                "ContainerPort": {
                                    "Ref": "ContainerPort"
                                },
                                "Protocol": "tcp"
                            }
                        ],
                        "Essential": true,
                        "LogConfiguration": {
                            "LogDriver": "awslogs",
                            "Options": {
                                "awslogs-group": {
                                    "Ref": "LogGroup"
                                },
```

```
                                "awslogs-region": {
                                    "Ref": "AWS::Region"
                                },
                                "awslogs-stream-prefix": "ecs"
                            }
                        },
                        "HealthCheck": {
                            "Command": [
                                "CMD-SHELL",
                                "curl -f http://localhost/ || exit 1"
                            ],
                            "Interval": 30,
                            "Timeout": 5,
                            "Retries": 3,
                            "StartPeriod": 60
                        }
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-task"
                        }
                    }
                ]
            }
        },
        "ECSService": {
            "Type": "AWS::ECS::Service",
            "DependsOn": "ALBListener",
            "Properties": {
                "ServiceName": {
                    "Fn::Sub": "${AWS::StackName}-service"
                },
                "Cluster": {
                    "Ref": "ECSCluster"
                },
                "TaskDefinition": {
                    "Ref": "TaskDefinition"
                },
                "DesiredCount": {
                    "Ref": "DesiredCount"
                },
```

```
                "LaunchType": "FARGATE",
                "PlatformVersion": "LATEST",
                "NetworkConfiguration": {
                    "AwsvpcConfiguration": {
                        "AssignPublicIp": "DISABLED",
                        "SecurityGroups": [
                            {
                                "Ref": "ECSSecurityGroup"
                            }
                        ],
                        "Subnets": [
                            {
                                "Ref": "PrivateSubnet1"
                            },
                            {
                                "Ref": "PrivateSubnet2"
                            }
                        ]
                    }
                },
                "LoadBalancers": [
                    {
                        "ContainerName": {
                            "Ref": "ServiceName"
                        },
                        "ContainerPort": {
                            "Ref": "ContainerPort"
                        },
                        "TargetGroupArn": {
                            "Ref": "ALBTargetGroup"
                        }
                    }
                ],
                "DeploymentConfiguration": {
                    "MaximumPercent": 200,
                    "MinimumHealthyPercent": 50,
                    "DeploymentCircuitBreaker": {
                        "Enable": true,
                        "Rollback": true
                    }
                },
                "EnableExecuteCommand": true,
                "Tags": [
                    {
```

```
                         "Key": "Name",
                         "Value": {
                             "Fn::Sub": "${AWS::StackName}-service"
                         }
                     }
                 ]
             }
         },
         "ServiceScalingTarget": {
             "Type": "AWS::ApplicationAutoScaling::ScalableTarget",
             "Properties": {
                 "MaxCapacity": {
                     "Ref": "MaxCapacity"
                 },
                 "MinCapacity": {
                     "Ref": "MinCapacity"
                 },
                 "ResourceId": {
                     "Fn::Sub": "service/${ECSCluster}/${ECSService.Name}"
                 },
                 "RoleARN": {
                     "Fn::Sub": "arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService"
                 },
                 "ScalableDimension": "ecs:service:DesiredCount",
                 "ServiceNamespace": "ecs"
             }
         },
         "ServiceScalingPolicy": {
             "Type": "AWS::ApplicationAutoScaling::ScalingPolicy",
             "Properties": {
                 "PolicyName": {
                     "Fn::Sub": "${AWS::StackName}-cpu-scaling-policy"
                 },
                 "PolicyType": "TargetTrackingScaling",
                 "ScalingTargetId": {
                     "Ref": "ServiceScalingTarget"
                 },
                 "TargetTrackingScalingPolicyConfiguration": {
                     "PredefinedMetricSpecification": {
                         "PredefinedMetricType": "ECSServiceAverageCPUUtilization"
                     },
                     "TargetValue": 70,
```

```
                    "ScaleOutCooldown": 300,
                    "ScaleInCooldown": 300
                }
            }
        }
    },
    "Outputs": {
        "VPCId": {
            "Description": "VPC ID",
            "Value": {
                "Ref": "VPC"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-VPC-ID"
                }
            }
        },
        "LoadBalancerURL": {
            "Description": "URL of the Application Load Balancer",
            "Value": {
                "Fn::Sub": "http://${ApplicationLoadBalancer.DNSName}"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-ALB-URL"
                }
            }
        },
        "ECSClusterName": {
            "Description": "Name of the ECS Cluster",
            "Value": {
                "Ref": "ECSCluster"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-ECS-Cluster"
                }
            }
        },
        "ECSServiceName": {
            "Description": "Name of the ECS Service",
            "Value": {
                "Fn::GetAtt": [
```

```
                    "ECSService",
                    "Name"
                ]
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-ECS-Service"
                }
            }
        },
        "PrivateSubnet1": {
            "Description": "Private Subnet 1 ID",
            "Value": {
                "Ref": "PrivateSubnet1"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-Private-Subnet-1"
                }
            }
        },
        "PrivateSubnet2": {
            "Description": "Private Subnet 2 ID",
            "Value": {
                "Ref": "PrivateSubnet2"
            },
            "Export": {
                "Name": {
                    "Fn::Sub": "${AWS::StackName}-Private-Subnet-2"
                }
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: '[AWSDocs] ECS: load-balanced-web-application'

Parameters:
  VpcCidr:
    Type: String
```

```
    Default: '10.0.0.0/16'
    Description: CIDR block for the VPC
  ContainerImage:
    Type: String
    Default: 'public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest'
    Description: Container image to use in task definition

  PublicSubnet1Cidr:
    Type: String
    Default: '10.0.1.0/24'
    Description: CIDR block for public subnet 1

  PublicSubnet2Cidr:
    Type: String
    Default: '10.0.2.0/24'
    Description: CIDR block for public subnet 2

  PrivateSubnet1Cidr:
    Type: String
    Default: '10.0.3.0/24'
    Description: CIDR block for private subnet 1

  PrivateSubnet2Cidr:
    Type: String
    Default: '10.0.4.0/24'
    Description: CIDR block for private subnet 2

  ServiceName:
    Type: String
    Default: 'tutorial-app'
    Description: Name of the ECS service

  ContainerPort:
    Type: Number
    Default: 80
    Description: Port on which the container listens

  DesiredCount:
    Type: Number
    Default: 2
    Description: Desired number of tasks

  MinCapacity:
    Type: Number
```

```
    Default: 1
    Description: Minimum number of tasks for auto scaling

  MaxCapacity:
    Type: Number
    Default: 10
    Description: Maximum number of tasks for auto scaling

Resources:
  # VPC and Networking
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCidr
      EnableDnsHostnames: true
      EnableDnsSupport: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-vpc'

  # Internet Gateway
  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-igw'

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  # Public Subnets for ALB
  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PublicSubnet1Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
```

```
          Value: !Sub '${AWS::StackName}-public-subnet-1'

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PublicSubnet2Cidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-subnet-2'

  # Private Subnets for ECS Tasks
  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet1Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-1'

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs '']
      CidrBlock: !Ref PrivateSubnet2Cidr
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-subnet-2'

  # NAT Gateways for private subnet internet access
  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-1'
```

```
  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-eip-2'

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-1'

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-nat-2'

  # Route Tables
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-public-routes'

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway
```

```
  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-1'

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-private-routes-2'

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
```

```
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  # Security Groups
  ALBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-alb-sg'
      GroupDescription: Security group for Application Load Balancer
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
          Description: Allow HTTP traffic from internet
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb-sg'

  ECSSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: !Sub '${AWS::StackName}-ecs-sg'
      GroupDescription: Security group for ECS tasks
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: !Ref ContainerPort
          ToPort: !Ref ContainerPort
          SourceSecurityGroupId: !Ref ALBSecurityGroup
          Description: Allow traffic from ALB
      SecurityGroupEgress:
```

```
        - IpProtocol: -1
          CidrIp: 0.0.0.0/0
          Description: Allow all outbound traffic
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-ecs-sg'

  # Application Load Balancer
  ApplicationLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Name: !Sub '${AWS::StackName}-alb'
      Scheme: internet-facing
      Type: application
      Subnets:
        - !Ref PublicSubnet1
        - !Ref PublicSubnet2
      SecurityGroups:
        - !Ref ALBSecurityGroup
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-alb'

  ALBTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Name: !Sub '${AWS::StackName}-tg'
      Port: !Ref ContainerPort
      Protocol: HTTP
      VpcId: !Ref VPC
      TargetType: ip
      HealthCheckIntervalSeconds: 30
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      UnhealthyThresholdCount: 5
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-tg'

  ALBListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
```

```
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref ALBTargetGroup
      LoadBalancerArn: !Ref ApplicationLoadBalancer
      Port: 80
      Protocol: HTTP

  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Sub '${AWS::StackName}-cluster'
      CapacityProviders:
        - FARGATE
        - FARGATE_SPOT
      DefaultCapacityProviderStrategy:
        - CapacityProvider: FARGATE
          Weight: 1
        - CapacityProvider: FARGATE_SPOT
          Weight: 4
      ClusterSettings:
        - Name: containerInsights
          Value: enabled
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-cluster'

  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: !Sub '${AWS::StackName}-task-execution-role'
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-task-execution-role'
```

```
ECSTaskRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: !Sub '${AWS::StackName}-task-role'
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: ecs-tasks.amazonaws.com
          Action: sts:AssumeRole
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-task-role'

# CloudWatch Log Group
LogGroup:
  Type: AWS::Logs::LogGroup
  Properties:
    LogGroupName: !Sub '/ecs/${AWS::StackName}'
    RetentionInDays: 7

# ECS Task Definition
TaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: !Sub '${AWS::StackName}-task'
    Cpu: '256'
    Memory: '512'
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - FARGATE
    ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
    TaskRoleArn: !GetAtt ECSTaskRole.Arn
    ContainerDefinitions:
      - Name: !Ref ServiceName
        Image: !Ref ContainerImage
        PortMappings:
          - ContainerPort: !Ref ContainerPort
            Protocol: tcp
        Essential: true
        LogConfiguration:
          LogDriver: awslogs
```

```
          Options:
            awslogs-group: !Ref LogGroup
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: ecs
        HealthCheck:
          Command:
            - CMD-SHELL
            - curl -f http://localhost/ || exit 1
          Interval: 30
          Timeout: 5
          Retries: 3
          StartPeriod: 60
    Tags:
      - Key: Name
        Value: !Sub '${AWS::StackName}-task'

# ECS Service
ECSService:
  Type: AWS::ECS::Service
  DependsOn: ALBListener
  Properties:
    ServiceName: !Sub '${AWS::StackName}-service'
    Cluster: !Ref ECSCluster
    TaskDefinition: !Ref TaskDefinition
    DesiredCount: !Ref DesiredCount
    LaunchType: FARGATE
    PlatformVersion: LATEST
    NetworkConfiguration:
      AwsvpcConfiguration:
        AssignPublicIp: DISABLED
        SecurityGroups:
          - !Ref ECSSecurityGroup
        Subnets:
          - !Ref PrivateSubnet1
          - !Ref PrivateSubnet2
    LoadBalancers:
      - ContainerName: !Ref ServiceName
        ContainerPort: !Ref ContainerPort
        TargetGroupArn: !Ref ALBTargetGroup
    DeploymentConfiguration:
      MaximumPercent: 200
      MinimumHealthyPercent: 50
      DeploymentCircuitBreaker:
        Enable: true
```

```
          Rollback: true
      EnableExecuteCommand: true  # For debugging
      Tags:
        - Key: Name
          Value: !Sub '${AWS::StackName}-service'

  # Auto Scaling Target
  ServiceScalingTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    Properties:
      MaxCapacity: !Ref MaxCapacity
      MinCapacity: !Ref MinCapacity
      ResourceId: !Sub 'service/${ECSCluster}/${ECSService.Name}'
      RoleARN: !Sub 'arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService'
      ScalableDimension: ecs:service:DesiredCount
      ServiceNamespace: ecs

  # Auto Scaling Policy - CPU Utilization
  ServiceScalingPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    Properties:
      PolicyName: !Sub '${AWS::StackName}-cpu-scaling-policy'
      PolicyType: TargetTrackingScaling
      ScalingTargetId: !Ref ServiceScalingTarget
      TargetTrackingScalingPolicyConfiguration:
        PredefinedMetricSpecification:
          PredefinedMetricType: ECSServiceAverageCPUUtilization
        TargetValue: 70.0
        ScaleOutCooldown: 300
        ScaleInCooldown: 300

Outputs:
  VPCId:
    Description: VPC ID
    Value: !Ref VPC
    Export:
      Name: !Sub '${AWS::StackName}-VPC-ID'

  LoadBalancerURL:
    Description: URL of the Application Load Balancer
    Value: !Sub 'http://${ApplicationLoadBalancer.DNSName}'
    Export:
```

```
      Name: !Sub '${AWS::StackName}-ALB-URL'

  ECSClusterName:
    Description: Name of the ECS Cluster
    Value: !Ref ECSCluster
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Cluster'

  ECSServiceName:
    Description: Name of the ECS Service
    Value: !GetAtt ECSService.Name
    Export:
      Name: !Sub '${AWS::StackName}-ECS-Service'

  PrivateSubnet1:
    Description: Private Subnet 1 ID
    Value: !Ref PrivateSubnet1
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-1'

  PrivateSubnet2:
    Description: Private Subnet 2 ID
    Value: !Ref PrivateSubnet2
    Export:
      Name: !Sub '${AWS::StackName}-Private-Subnet-2'
```

Deploy a service with ECS Exec enabled

You can use the following template to deploy a service with ECS Exec enabled. The service runs in a cluster with a KMS key for encrypting ECS Exec sessions and a log configuration for redirecting execute command session logs to an Amazon S3 bucket. For more information, see [Monitor Amazon ECS containers with ECS Exec](#).

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "ECS Cluster with Fargate Service and Task Definition and ECS
 Exec enabled.",
    "Parameters": {
        "ClusterName": {
            "Type": "String",
```

```
            "Default": "CFNCluster",
            "Description": "Name of the ECS Cluster"
        },
        "S3BucketName": {
            "Type": "String",
            "Default": "amzn-s3-demo-bucket",
            "Description": "S3 bucket for ECS execute command logs"
        },
        "KmsKeyId": {
            "Type": "String",
            "Default": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "Description": "KMS Key ID for ECS execute command encryption"
        },
        "ContainerImage": {
            "Type": "String",
            "Default": "public.ecr.aws/docker/library/httpd:2.4",
            "Description": "Container image to use for the task"
        },
        "ContainerCpu": {
            "Type": "Number",
            "Default": 256,
            "AllowedValues": [256, 512, 1024, 2048, 4096],
            "Description": "CPU units for the container (256 = 0.25 vCPU)"
        },
        "ContainerMemory": {
            "Type": "Number",
            "Default": 512,
            "AllowedValues": [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192],
            "Description": "Memory for the container (in MiB)"
        },
        "DesiredCount": {
            "Type": "Number",
            "Default": 1,
            "Description": "Desired count of tasks in the service"
        },
       "SecurityGroups": {
   "Type": "List<AWS::EC2::SecurityGroup::Id>",
   "Description": "Security Group IDs for the ECS Service"
  },
  "Subnets": {
   "Type": "List<AWS::EC2::Subnet::Id>",
   "Description": "Subnet IDs for the ECS Service"
  },
        "ServiceName": {
```

```
            "Type": "String",
            "Default": "cfn-service",
            "Description": "Name of the ECS service"
        },
        "TaskFamily": {
            "Type": "String",
            "Default": "task-definition-cfn",
            "Description": "Family name for the task definition"
        },
        "TaskExecutionRoleArn": {
            "Type": "String",
            "Description": "ARN of an existing IAM role for ECS task execution",
            "Default": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
        },
        "TaskRoleArn": {
            "Type": "String",
            "Description": "ARN of an existing IAM role for ECS tasks",
            "Default": "arn:aws:iam::111122223333:role/execTaskRole"
        }
    },
    "Resources": {
        "ECSCluster": {
            "Type": "AWS::ECS::Cluster",
            "Properties": {
                "ClusterName": {"Ref": "ClusterName"},
                "Configuration": {
                    "ExecuteCommandConfiguration": {
                        "Logging": "OVERRIDE",
                        "LogConfiguration": {
                            "S3BucketName": {"Ref": "S3BucketName"}
                        },
                        "KmsKeyId": {"Ref": "KmsKeyId"}
                    }
                },
                "Tags": [
                    {
                        "Key": "Environment",
                        "Value": {"Ref": "AWS::StackName"}
                    }
                ]
            }
        },
        "ECSTaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
```

```
            "Properties": {
                "ContainerDefinitions": [
                    {
                        "Command": [
                            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS
 Sample App</title> <style>body {margin-top: 40px; background-color: #333;} </style>
 </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
                        ],
                        "EntryPoint": [
                            "sh",
                            "-c"
                        ],
                        "Essential": true,
                        "Image": {"Ref": "ContainerImage"},
                        "LogConfiguration": {
                            "LogDriver": "awslogs",
                            "Options": {
                                "mode": "non-blocking",
                                "max-buffer-size": "25m",
                                "awslogs-create-group": "true",
                                "awslogs-group": {"Fn::Sub": "/ecs/
${AWS::StackName}"},
                                "awslogs-region": {"Ref": "AWS::Region"},
                                "awslogs-stream-prefix": "ecs"
                            }
                        },
                        "Name": "sample-fargate-app",
                        "PortMappings": [
                            {
                                "ContainerPort": 80,
                                "HostPort": 80,
                                "Protocol": "tcp"
                            }
                        ]
                    }
                ],
                "Cpu": {"Ref": "ContainerCpu"},
                "ExecutionRoleArn": {"Ref": "TaskExecutionRoleArn"},
                "TaskRoleArn": {"Ref": "TaskRoleArn"},
                "Family": {"Ref": "TaskFamily"},
                "Memory": {"Ref": "ContainerMemory"},
```

```
                "NetworkMode": "awsvpc",
                "RequiresCompatibilities": [
                    "FARGATE"
                ],
                "RuntimePlatform": {
                    "OperatingSystemFamily": "LINUX"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {"Fn::Sub": "${AWS::StackName}-TaskDefinition"}
                    }
                ]
            }
        },
        "ECSService": {
            "Type": "AWS::ECS::Service",
            "Properties": {
                "ServiceName": {"Ref": "ServiceName"},
                "Cluster": {"Ref": "ECSCluster"},
                "DesiredCount": {"Ref": "DesiredCount"},
                "EnableExecuteCommand": true,
                "LaunchType": "FARGATE",
                "NetworkConfiguration": {
                    "AwsvpcConfiguration": {
                        "AssignPublicIp": "ENABLED",
                        "SecurityGroups": {"Ref": "SecurityGroups"},
                        "Subnets": {"Ref": "Subnets"}
                    }
                },
                "TaskDefinition": {"Ref": "ECSTaskDefinition"},
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {"Fn::Sub": "${AWS::StackName}-Service"}
                    }
                ]
            }
        }
    },
    "Outputs": {
        "ClusterName": {
            "Description": "The name of the ECS cluster",
            "Value": {"Ref": "ECSCluster"}
```

```
        },
        "ServiceName": {
            "Description": "The name of the ECS service",
            "Value": {"Ref": "ServiceName"}
        },
        "TaskDefinitionArn": {
            "Description": "The ARN of the task definition",
            "Value": {"Ref": "ECSTaskDefinition"}
        },
        "ClusterArn": {
            "Description": "The ARN of the ECS cluster",
            "Value": {"Fn::GetAtt": ["ECSCluster", "Arn"]}
        },
        "StackName": {
            "Description": "The name of this stack",
            "Value": {"Ref": "AWS::StackName"}
        },
        "StackId": {
            "Description": "The unique identifier for this stack",
            "Value": {"Ref": "AWS::StackId"}
        },
        "Region": {
            "Description": "The AWS Region where the stack is deployed",
            "Value": {"Ref": "AWS::Region"}
        },
        "AccountId": {
            "Description": "The AWS Account ID",
            "Value": {"Ref": "AWS::AccountId"}
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: ECS Cluster with Fargate Service and Task Definition and ECS Exec
 enabled.
Parameters:
  ClusterName:
    Type: String
    Default: CFNCluster
    Description: Name of the ECS Cluster
  S3BucketName:
```

```
    Type: String
    Default: amzn-s3-demo-bucket
    Description: S3 bucket for ECS execute command logs
  KmsKeyId:
    Type: String
    Default: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
    Description: KMS Key ID for ECS execute command encryption
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  ContainerCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
    Description: CPU units for the container (256 = 0.25 vCPU)
  ContainerMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192]
    Description: Memory for the container (in MiB)
  DesiredCount:
    Type: Number
    Default: 1
    Description: Desired count of tasks in the service
 SecurityGroups:
    Type: List<AWS::EC2::SecurityGroup::Id>
    Description: Security Group IDs for the ECS Service
  Subnets:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service
  ServiceName:
    Type: String
    Default: cfn-service
    Description: Name of the ECS service
  TaskFamily:
    Type: String
    Default: task-definition-cfn
    Description: Family name for the task definition
  TaskExecutionRoleArn:
    Type: String
    Description: ARN of an existing IAM role for ECS task execution
    Default: 'arn:aws:iam::111122223333:role/ecsTaskExecutionRole'
  TaskRoleArn:
```

```
    Type: String
    Description: ARN of an existing IAM role for ECS tasks
    Default: 'arn:aws:iam::111122223333:role/execTaskRole'
Resources:
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ClusterName
      Configuration:
        ExecuteCommandConfiguration:
          Logging: OVERRIDE
          LogConfiguration:
            S3BucketName: !Ref S3BucketName
          KmsKeyId: !Ref KmsKeyId
      Tags:
        - Key: Environment
          Value: !Ref AWS::StackName
  ECSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
             /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
              App</title> <style>body {margin-top: 40px; background-color:
              #333;} </style> </head><body> <div
              style=color:white;text-align:center> <h1>Amazon ECS Sample
              App</h1> <h2>Congratulations!</h2> <p>Your application is now
              running on a container in Amazon ECS.</p> </div></body></html>' >
              /usr/local/apache2/htdocs/index.html && httpd-foreground"
          EntryPoint:
            - sh
            - '-c'
          Essential: true
          Image: !Ref ContainerImage
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
              awslogs-create-group: 'true'
              awslogs-group: !Sub /ecs/${AWS::StackName}
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: ecs
```

```
          Name: sample-fargate-app
          PortMappings:
            - ContainerPort: 80
              HostPort: 80
              Protocol: tcp
      Cpu: !Ref ContainerCpu
      ExecutionRoleArn: !Ref TaskExecutionRoleArn
      TaskRoleArn: !Ref TaskRoleArn
      Family: !Ref TaskFamily
      Memory: !Ref ContainerMemory
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      RuntimePlatform:
        OperatingSystemFamily: LINUX
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-TaskDefinition
  ECSService:
    Type: AWS::ECS::Service
    Properties:
      ServiceName: !Ref ServiceName
      Cluster: !Ref ECSCluster
      DesiredCount: !Ref DesiredCount
      EnableExecuteCommand: true
      LaunchType: FARGATE
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroups
          Subnets: !Ref Subnets
      TaskDefinition: !Ref ECSTaskDefinition
      Tags:
        - Key: Name
          Value: !Sub ${AWS::StackName}-Service
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref ECSCluster
  ServiceName:
    Description: The name of the ECS service
    Value: !Ref ServiceName
  TaskDefinitionArn:
    Description: The ARN of the task definition
```

```
    Value: !Ref ECSTaskDefinition
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  StackName:
    Description: The name of this stack
    Value: !Ref AWS::StackName
  StackId:
    Description: The unique identifier for this stack
    Value: !Ref AWS::StackId
  Region:
    Description: The AWS Region where the stack is deployed
    Value: !Ref AWS::Region
  AccountId:
    Description: The AWS Account ID
    Value: !Ref AWS::AccountId
```

Deploy service that uses Amazon VPC Lattice

You can use the following template to get started with creating an Amazon ECS service with VPC Lattice. You may need to complete the following additional steps to set up VPC Lattice:

- Update your security group's inbound rules for VPC Lattice to allow the inbound rule `vpc-lattice` prefix and to allow traffic on port 80.
- Associate VPC for the service to a VPC Lattice service network.
- Configure a private or public hosted zone with Amazon Route 53.
- Configure listeners and listener rules in a VPC Lattice service.
- Verify health check configurations of the target group.

For more information about using VPC Lattice with Amazon ECS, see [Use Amazon VPC Lattice to connect, observe, and secure your Amazon ECS services](#).

JSON

```
{
 "AWSTemplateFormatVersion": "2010-09-09",
 "Description": "The template used to create an ECS Service with VPC Lattice.",
 "Parameters": {
  "ECSClusterName": {
```

```
    "Type": "String",
    "Default": "vpc-lattice-cluster"
   },
   "ECSServiceName": {
    "Type": "String",
    "Default": "vpc-lattice-service"
   },
   "SecurityGroupIDs": {
    "Type": "List<AWS::EC2::SecurityGroup::Id>",
    "Description": "Security Group IDs for the ECS Service"
   },
   "SubnetIDs": {
    "Type": "List<AWS::EC2::Subnet::Id>",
    "Description": "Subnet IDs for the ECS Service"
   },
   "VpcID": {
    "Type": "AWS::EC2::VPC::Id",
    "Description": "VPC ID for the resources"
   },
   "ContainerImage": {
    "Type": "String",
    "Default": "public.ecr.aws/docker/library/httpd:2.4",
    "Description": "Container image to use for the task"
   },
   "TaskCpu": {
    "Type": "Number",
    "Default": 256,
    "AllowedValues": [256, 512, 1024, 2048, 4096],
    "Description": "CPU units for the task"
   },
   "TaskMemory": {
    "Type": "Number",
    "Default": 512,
    "AllowedValues": [512, 1024, 2048, 4096, 8192, 16384],
    "Description": "Memory (in MiB) for the task"
   },
   "LogGroupName": {
    "Type": "String",
    "Default": "/ecs/vpc-lattice-task",
    "Description": "CloudWatch Log Group name"
   },
   "EnableContainerInsights": {
    "Type": "String",
    "Default": "enabled",
```

```
    "AllowedValues": ["enabled", "disabled"],
    "Description": "Enable or disable CloudWatch Container Insights for the cluster"
   }
  },
  "Resources": {
   "ECSCluster": {
    "Type": "AWS::ECS::Cluster",
    "Properties": {
     "ClusterName": {"Ref": "ECSClusterName"},
     "ClusterSettings": [
      {
       "Name": "containerInsights",
       "Value": {"Ref": "EnableContainerInsights"}
      }
     ],
     "Tags": [
      {
       "Key": "Name",
       "Value": {"Ref": "ECSClusterName"}
      }
     ]
    }
   },
   "ECSTaskExecutionRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
     "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
       {
        "Effect": "Allow",
        "Principal": {
         "Service": "ecs-tasks.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
       }
      ]
     },
     "ManagedPolicyArns": [
      "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
     ]
    }
   },
   "TaskLogGroup": {
```

```
    "Type": "AWS::Logs::LogGroup",
    "DeletionPolicy": "Retain",
    "UpdateReplacePolicy": "Retain",
    "Properties": {
     "LogGroupName": {"Ref": "LogGroupName"},
     "RetentionInDays": 30
    }
   },
   "VpcLatticeTaskDefinition": {
    "Type": "AWS::ECS::TaskDefinition",
    "Properties": {
     "ContainerDefinitions": [
      {
       "Command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
<style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
<div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
<h2>Congratulations!</h2> <p>Your application is now running on a container in
Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
httpd-foreground\""
       ],
       "EntryPoint": [
        "sh",
        "-c"
       ],
       "Essential": true,
       "Image": {"Ref": "ContainerImage"},
       "LogConfiguration": {
        "LogDriver": "awslogs",
        "Options": {
         "mode": "non-blocking",
         "max-buffer-size": "25m",
         "awslogs-create-group": "true",
         "awslogs-group": {"Ref": "LogGroupName"},
         "awslogs-region": {"Ref": "AWS::Region"},
         "awslogs-stream-prefix": "ecs"
        }
       },
       "Name": "vpc-lattice-container",
       "PortMappings": [
        {
         "ContainerPort": 80,
         "HostPort": 80,
         "Protocol": "tcp",
```

```
          "Name": "vpc-lattice-port"
        }
       ]
      }
     ],
     "Cpu": {"Ref": "TaskCpu"},
     "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
     "Family": "vpc-lattice-task-definition",
     "Memory": {"Ref": "TaskMemory"},
     "NetworkMode": "awsvpc",
     "RequiresCompatibilities": [
      "FARGATE"
     ],
     "RuntimePlatform": {
      "OperatingSystemFamily": "LINUX"
     }
    }
   },
   "ECSService": {
    "Type": "AWS::ECS::Service",
    "Properties": {
     "Cluster": {"Ref": "ECSCluster"},
     "TaskDefinition": {"Ref": "VpcLatticeTaskDefinition"},
     "LaunchType": "FARGATE",
     "ServiceName": {"Ref": "ECSServiceName"},
     "SchedulingStrategy": "REPLICA",
     "DesiredCount": 2,
     "AvailabilityZoneRebalancing": "ENABLED",
     "NetworkConfiguration": {
      "AwsvpcConfiguration": {
       "AssignPublicIp": "ENABLED",
       "SecurityGroups": {
        "Ref": "SecurityGroupIDs"
       },
       "Subnets": {
        "Ref": "SubnetIDs"
       }
      }
     },
     "PlatformVersion": "LATEST",
     "VpcLatticeConfigurations": [
      {
       "RoleArn": "arn:aws:iam::111122223333:role/ecsInfrastructureRole",
       "PortName": "vpc-lattice-port",
```

```
        "TargetGroupArn": {
         "Ref": "TargetGroup1"
        }
       }
      ],
      "DeploymentConfiguration": {
       "DeploymentCircuitBreaker": {
        "Enable": true,
        "Rollback": true
       },
       "MaximumPercent": 200,
       "MinimumHealthyPercent": 100
      },
      "DeploymentController": {
       "Type": "ECS"
      },
      "ServiceConnectConfiguration": {
       "Enabled": false
      },
      "Tags": [],
      "EnableECSManagedTags": true
     }
    },
    "TargetGroup1": {
     "Type": "AWS::VpcLattice::TargetGroup",
     "Properties": {
      "Type": "IP",
      "Name": "first-target-group",
      "Config": {
       "Port": 80,
       "Protocol": "HTTP",
       "VpcIdentifier": {"Ref": "VpcID"},
       "HealthCheck": {
        "Enabled": true,
        "Path": "/"
       }
      },
      "Tags": [
       {
        "Key": "ecs-application-networking/ServiceName",
        "Value": {"Ref": "ECSServiceName"}
       },
       {
        "Key": "ecs-application-networking/ClusterName",
```

```
          "Value": {"Ref": "ECSClusterName"}
        },
        {
          "Key": "ecs-application-networking/TaskDefinition",
          "Value": {"Ref": "VpcLatticeTaskDefinition"}
        },
        {
          "Key": "ecs-application-networking/VpcId",
          "Value": {"Ref": "VpcID"}
        }
      ]
    }
  }
 },
 "Outputs": {
  "ClusterName": {
   "Description": "The cluster used to create the service.",
   "Value": {
    "Ref": "ECSCluster"
   }
  },
  "ClusterArn": {
   "Description": "The ARN of the ECS cluster",
   "Value": {
    "Fn::GetAtt": ["ECSCluster", "Arn"]
   }
  },
  "ECSService": {
   "Description": "The created service.",
   "Value": {
    "Ref": "ECSService"
   }
  },
  "TaskDefinitionArn": {
   "Description": "The ARN of the task definition",
   "Value": {
    "Ref": "VpcLatticeTaskDefinition"
   }
  }
 }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: The template used to create an ECS Service with VPC Lattice.

Parameters:
  ECSClusterName:
    Type: String
    Default: vpc-lattice-cluster
  ECSServiceName:
    Type: String
    Default: vpc-lattice-service
  SecurityGroupIDs:
    Type: List<AWS::EC2::SecurityGroup::Id>
    Description: Security Group IDs for the ECS Service
  SubnetIDs:
    Type: List<AWS::EC2::Subnet::Id>
    Description: Subnet IDs for the ECS Service
  VpcID:
    Type: AWS::EC2::VPC::Id
    Description: VPC ID for the resources
  ContainerImage:
    Type: String
    Default: public.ecr.aws/docker/library/httpd:2.4
    Description: Container image to use for the task
  TaskCpu:
    Type: Number
    Default: 256
    AllowedValues: [256, 512, 1024, 2048, 4096]
    Description: CPU units for the task
  TaskMemory:
    Type: Number
    Default: 512
    AllowedValues: [512, 1024, 2048, 4096, 8192, 16384]
    Description: Memory (in MiB) for the task
  LogGroupName:
    Type: String
    Default: /ecs/vpc-lattice-task
    Description: CloudWatch Log Group name
  EnableContainerInsights:
    Type: String
    Default: 'enhanced'
    AllowedValues: ['enabled', 'disabled', 'enhanced']
    Description: Enable or disable CloudWatch Container Insights for the cluster
```

```
Resources:
  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ECSClusterName
      ClusterSettings:
        - Name: containerInsights
          Value: !Ref EnableContainerInsights
      Tags:
        - Key: Name
          Value: !Ref ECSClusterName

  # IAM Roles
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

  # CloudWatch Logs
  TaskLogGroup:
    Type: AWS::Logs::LogGroup
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      LogGroupName: !Ref LogGroupName
      RetentionInDays: 30

  # Task Definition
  VpcLatticeTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
        - Command:
            - >-
```

```
            /bin/sh -c "echo '<html> <head> <title>Amazon ECS Sample
            App</title> <style>body {margin-top: 40px; background-color:
            #333;} </style> </head><body> <div
            style=color:white;text-align:center> <h1>Amazon ECS Sample
            App</h1> <h2>Congratulations!</h2> <p>Your application is now
            running on a container in Amazon ECS.</p> </div></body></html>' >
            /usr/local/apache2/htdocs/index.html && httpd-foreground"
        EntryPoint:
          - sh
          - '-c'
        Essential: true
        Image: !Ref ContainerImage
        LogConfiguration:
          LogDriver: awslogs
          Options:
            mode: non-blocking
            max-buffer-size: 25m
            awslogs-create-group: 'true'
            awslogs-group: !Ref LogGroupName
            awslogs-region: !Ref 'AWS::Region'
            awslogs-stream-prefix: ecs
        Name: vpc-lattice-container
        PortMappings:
          - ContainerPort: 80
            HostPort: 80
            Protocol: tcp
            Name: vpc-lattice-port
    Cpu: !Ref TaskCpu
    ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
    Family: vpc-lattice-task-definition
    Memory: !Ref TaskMemory
    NetworkMode: awsvpc
    RequiresCompatibilities:
      - FARGATE
    RuntimePlatform:
      OperatingSystemFamily: LINUX

ECSService:
  Type: AWS::ECS::Service
  Properties:
    Cluster: !Ref ECSCluster
    TaskDefinition: !Ref VpcLatticeTaskDefinition
    LaunchType: FARGATE
    ServiceName: !Ref ECSServiceName
```

```
      SchedulingStrategy: REPLICA
      DesiredCount: 2
      AvailabilityZoneRebalancing: ENABLED
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroupIDs
          Subnets: !Ref SubnetIDs
      PlatformVersion: LATEST
      VpcLatticeConfigurations:
        - RoleArn: arn:aws:iam::111122223333:role/ecsInfrastructureRole
          PortName: vpc-lattice-port
          TargetGroupArn: !Ref TargetGroup1
      DeploymentConfiguration:
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
        MaximumPercent: 200
        MinimumHealthyPercent: 100
      DeploymentController:
        Type: ECS
      ServiceConnectConfiguration:
        Enabled: false
      Tags: []
      EnableECSManagedTags: true

  TargetGroup1:
    Type: AWS::VpcLattice::TargetGroup
    Properties:
      Type: IP
      Name: first-target-group
      Config:
        Port: 80
        Protocol: HTTP
        VpcIdentifier: !Ref VpcID
        HealthCheck:
          Enabled: true
          Path: /
      Tags:
        - Key: ecs-application-networking/ServiceName
          Value: !Ref ECSServiceName
        - Key: ecs-application-networking/ClusterName
          Value: !Ref ECSClusterName
        - Key: ecs-application-networking/TaskDefinition
```

```
          Value: !Ref VpcLatticeTaskDefinition
        - Key: ecs-application-networking/VpcId
          Value: !Ref VpcID

Outputs:
  ClusterName:
    Description: The cluster used to create the service.
    Value: !Ref ECSCluster
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  ECSService:
    Description: The created service.
    Value: !Ref ECSService
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref VpcLatticeTaskDefinition
```

Deploy service with a volume configuration

The following template includes a volume configuration in the service definition. Amazon ECS supports configuring the following data volumes by using a volume configuration at launch: Amazon EBS volumes. For more information about Amazon EBS volumes, see Use Amazon EBS volumes with Amazon ECS.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "The template used to create an ECS Service that includes a
 volume configuration. The configuration is used to create Amazon EBS volumes for
 attachment to the tasks. One volume is attached per task.",
  "Parameters": {
    "ECSClusterName": {
      "Type": "String",
      "Default": "volume-config-cluster",
      "Description": "Name of the ECS cluster"
    },
    "SecurityGroupIDs": {
      "Type": "List<AWS::EC2::SecurityGroup::Id>",
      "Description": "Security Group IDs for the ECS Service"
    },
```

```
    "SubnetIDs": {
      "Type": "List<AWS::EC2::Subnet::Id>",
      "Description": "Subnet IDs for the ECS Service"
    },
    "InfrastructureRoleArn": {
      "Type": "String",
      "Description": "ARN of the IAM role that ECS will use to manage EBS volumes"
    },
    "ContainerImage": {
      "Type": "String",
      "Default": "public.ecr.aws/nginx/nginx:latest",
      "Description": "Container image to use for the task"
    },
    "TaskCpu": {
      "Type": "String",
      "Default": "2048",
      "Description": "CPU units for the task"
    },
    "TaskMemory": {
      "Type": "String",
      "Default": "4096",
      "Description": "Memory (in MiB) for the task"
    },
    "VolumeSize": {
      "Type": "String",
      "Default": "10",
      "Description": "Size of the EBS volume in GiB"
    },
    "VolumeType": {
      "Type": "String",
      "Default": "gp3",
      "AllowedValues": ["gp2", "gp3", "io1", "io2", "st1", "sc1", "standard"],
      "Description": "EBS volume type"
    },
    "VolumeIops": {
      "Type": "String",
      "Default": "3000",
      "Description": "IOPS for the EBS volume (required for io1, io2, and gp3)"
    },
    "VolumeThroughput": {
      "Type": "String",
      "Default": "125",
      "Description": "Throughput for the EBS volume (only for gp3)"
    },
```

```
    "FilesystemType": {
      "Type": "String",
      "Default": "xfs",
      "AllowedValues": ["xfs", "ext4"],
      "Description": "Filesystem type for the EBS volume"
    },
    "EnableContainerInsights": {
      "Type": "String",
      "Default": "enhanced",
      "AllowedValues": ["enabled", "disabled", "enhanced"],
      "Description": "Enable or disable CloudWatch Container Insights for the
  cluster"
    }
  },
  "Resources": {
    "ECSCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {
        "ClusterName": {"Ref": "ECSClusterName"},
        "ClusterSettings": [
          {
            "Name": "containerInsights",
            "Value": {"Ref": "EnableContainerInsights"}
          }
        ],
        "Tags": [
          {
            "Key": "Name",
            "Value": {"Ref": "ECSClusterName"}
          }
        ]
      }
    },
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
```

```
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    },
    "EBSTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "Family": "ebs-task-attach-task-def",
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "NetworkMode": "awsvpc",
        "RequiresCompatibilities": [
          "EC2",
          "FARGATE"
        ],
        "Cpu": {"Ref": "TaskCpu"},
        "Memory": {"Ref": "TaskMemory"},
        "ContainerDefinitions": [
          {
            "Name": "nginx",
            "Image": {"Ref": "ContainerImage"},
            "Essential": true,
            "PortMappings": [
              {
                "Name": "nginx-80-tcp",
                "ContainerPort": 80,
                "HostPort": 80,
                "Protocol": "tcp",
                "AppProtocol": "http"
              }
            ],
            "MountPoints": [
              {
                "SourceVolume": "ebs-vol",
                "ContainerPath": "/foo-container-path",
                "ReadOnly": false
              }
            ]
          }
        ],
```

```
        "Volumes": [
          {
            "Name": "ebs-vol",
            "ConfiguredAtLaunch": true
          }
        ]
      }
    },
    "ECSService": {
      "Type": "AWS::ECS::Service",
      "Properties": {
        "Cluster": {"Ref": "ECSCluster"},
        "TaskDefinition": {"Ref": "EBSTaskDefinition"},
        "LaunchType": "FARGATE",
        "ServiceName": "ebs",
        "SchedulingStrategy": "REPLICA",
        "DesiredCount": 1,
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": {"Ref": "SecurityGroupIDs"},
            "Subnets": {"Ref": "SubnetIDs"}
          }
        },
        "PlatformVersion": "LATEST",
        "DeploymentConfiguration": {
          "MaximumPercent": 200,
          "MinimumHealthyPercent": 100,
          "DeploymentCircuitBreaker": {
            "Enable": true,
            "Rollback": true
          }
        },
        "DeploymentController": {
          "Type": "ECS"
        },
        "Tags": [],
        "EnableECSManagedTags": true,
        "VolumeConfigurations": [
          {
            "Name": "ebs-vol",
            "ManagedEBSVolume": {
              "RoleArn": {"Ref": "InfrastructureRoleArn"},
              "VolumeType": {"Ref": "VolumeType"},
```

```
                "Iops": {"Ref": "VolumeIops"},
                "Throughput": {"Ref": "VolumeThroughput"},
                "SizeInGiB": {"Ref": "VolumeSize"},
                "FilesystemType": {"Ref": "FilesystemType"},
                "TagSpecifications": [
                  {
                    "ResourceType": "volume",
                    "PropagateTags": "TASK_DEFINITION"
                  }
                ]
              }
            }
          ]
        }
      }
    },
    "Outputs": {
      "ClusterName": {
        "Description": "The cluster used to create the service.",
        "Value": {"Ref": "ECSCluster"}
      },
      "ClusterArn": {
        "Description": "The ARN of the ECS cluster",
        "Value": {"Fn::GetAtt": ["ECSCluster", "Arn"]}
      },
      "ECSService": {
        "Description": "The created service.",
        "Value": {"Ref": "ECSService"}
      },
      "TaskDefinitionArn": {
        "Description": "The ARN of the task definition",
        "Value": {"Ref": "EBSTaskDefinition"}
      }
    }
  }
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: The template used to create an ECS Service that includes a volume
 configuration. The configuration is used to create Amazon EBS volumes for
 attachment to the tasks. One volume is attached per task.
Parameters:
```

```
ECSClusterName:
  Type: String
  Default: volume-config-cluster
  Description: Name of the ECS cluster

SecurityGroupIDs:
  Type: List<AWS::EC2::SecurityGroup::Id>
  Description: Security Group IDs for the ECS Service

SubnetIDs:
  Type: List<AWS::EC2::Subnet::Id>
  Description: Subnet IDs for the ECS Service

InfrastructureRoleArn:
  Type: String
  Description: ARN of the IAM role that ECS will use to manage EBS volumes

ContainerImage:
  Type: String
  Default: public.ecr.aws/nginx/nginx:latest
  Description: Container image to use for the task

TaskCpu:
  Type: String
  Default: "2048"
  Description: CPU units for the task

TaskMemory:
  Type: String
  Default: "4096"
  Description: Memory (in MiB) for the task

VolumeSize:
  Type: String
  Default: "10"
  Description: Size of the EBS volume in GiB

VolumeType:
  Type: String
  Default: gp3
  AllowedValues: [gp2, gp3, io1, io2, st1, sc1, standard]
  Description: EBS volume type

VolumeIops:
```

```
    Type: String
    Default: "3000"
    Description: IOPS for the EBS volume (required for io1, io2, and gp3)

  VolumeThroughput:
    Type: String
    Default: "125"
    Description: Throughput for the EBS volume (only for gp3)

  FilesystemType:
    Type: String
    Default: xfs
    AllowedValues: [xfs, ext4]
    Description: Filesystem type for the EBS volume

  EnableContainerInsights:
    Type: String
    Default: 'enhanced'
    AllowedValues: ['enabled', 'disabled', 'enhanced']
    Description: Enable or disable CloudWatch Container Insights for the cluster

Resources:
  # ECS Cluster
  ECSCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref ECSClusterName
      ClusterSettings:
        - Name: containerInsights
          Value: !Ref EnableContainerInsights
      Tags:
        - Key: Name
          Value: !Ref ECSClusterName

  # IAM Role for Task Execution
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ecs-tasks.amazonaws.com
```

```
          Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

  # Task Definition
  EBSTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: ebs-task-attach-task-def
      ExecutionRoleArn: !GetAtt ECSTaskExecutionRole.Arn
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - EC2
        - FARGATE
      Cpu: !Ref TaskCpu
      Memory: !Ref TaskMemory
      ContainerDefinitions:
        - Name: nginx
          Image: !Ref ContainerImage
          Essential: true
          PortMappings:
            - Name: nginx-80-tcp
              ContainerPort: 80
              HostPort: 80
              Protocol: tcp
              AppProtocol: http
          MountPoints:
            - SourceVolume: ebs-vol
              ContainerPath: /foo-container-path
              ReadOnly: false
      Volumes:
        - Name: ebs-vol
          ConfiguredAtLaunch: true

  ECSService:
    Type: AWS::ECS::Service
    Properties:
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref EBSTaskDefinition
      LaunchType: FARGATE
      ServiceName: ebs
      SchedulingStrategy: REPLICA
      DesiredCount: 1
      NetworkConfiguration:
```

```
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups: !Ref SecurityGroupIDs
          Subnets: !Ref SubnetIDs
      PlatformVersion: LATEST
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 100
        DeploymentCircuitBreaker:
          Enable: true
          Rollback: true
      DeploymentController:
        Type: ECS
      Tags: []
      EnableECSManagedTags: true
      VolumeConfigurations:
        - Name: ebs-vol
          ManagedEBSVolume:
            RoleArn: !Ref InfrastructureRoleArn
            VolumeType: !Ref VolumeType
            Iops: !Ref VolumeIops
            Throughput: !Ref VolumeThroughput
            SizeInGiB: !Ref VolumeSize
            FilesystemType: !Ref FilesystemType
            TagSpecifications:
              - ResourceType: volume
                PropagateTags: TASK_DEFINITION

Outputs:
  ClusterName:
    Description: The cluster used to create the service.
    Value: !Ref ECSCluster
  ClusterArn:
    Description: The ARN of the ECS cluster
    Value: !GetAtt ECSCluster.Arn
  ECSService:
    Description: The created service.
    Value: !Ref ECSService
  TaskDefinitionArn:
    Description: The ARN of the task definition
    Value: !Ref EBSTaskDefinition
```

Deploy service with capacity providers

The following template defines a service that uses the capacity provider to request AL2023 capacity to run on. Containers will be launched onto the AL2023 instances as they come online.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "An example service that deploys in AWS VPC networking mode on
 EC2 capacity. Service uses a capacity provider to request EC2 instances to run
 on. Service runs with networking in private subnets, but still accessible to the
 internet via a load balancer hosted in public subnets.",
  "Parameters": {
      "VpcId": {
          "Type": "String",
          "Description": "The VPC that the service is running inside of"
      },
      "PublicSubnetIds": {
          "Type": "List<AWS::EC2::Subnet::Id>",
          "Description": "List of public subnet ID's to put the load balancer in"
      },
      "PrivateSubnetIds": {
          "Type": "List<AWS::EC2::Subnet::Id>",
          "Description": "List of private subnet ID's that the AWS VPC tasks are in"
      },
      "ClusterName": {
          "Type": "String",
          "Description": "The name of the ECS cluster into which to launch
 capacity."
      },
      "ECSTaskExecutionRole": {
          "Type": "String",
          "Description": "The role used to start up an ECS task"
      },
      "CapacityProvider": {
          "Type": "String",
          "Description": "The cluster capacity provider that the service should use
 to request capacity when it wants to start up a task"
      },
      "ServiceName": {
          "Type": "String",
          "Default": "web",
```

```
            "Description": "A name for the service"
        },
        "ImageUrl": {
            "Type": "String",
            "Default": "public.ecr.aws/docker/library/nginx:latest",
            "Description": "The url of a docker image that contains the application
 process that will handle the traffic for this service"
        },
        "ContainerCpu": {
            "Type": "Number",
            "Default": 256,
            "Description": "How much CPU to give the container. 1024 is 1 CPU"
        },
        "ContainerMemory": {
            "Type": "Number",
            "Default": 512,
            "Description": "How much memory in megabytes to give the container"
        },
        "ContainerPort": {
            "Type": "Number",
            "Default": 80,
            "Description": "What port that the application expects traffic on"
        },
        "DesiredCount": {
            "Type": "Number",
            "Default": 2,
            "Description": "How many copies of the service task to run"
        }
    },
    "Resources": {
        "TaskDefinition": {
            "Type": "AWS::ECS::TaskDefinition",
            "Properties": {
                "Family": {
                    "Ref": "ServiceName"
                },
                "Cpu": {
                    "Ref": "ContainerCpu"
                },
                "Memory": {
                    "Ref": "ContainerMemory"
                },
                "NetworkMode": "awsvpc",
                "RequiresCompatibilities": [
```

```
                "EC2"
            ],
            "ExecutionRoleArn": {
                "Ref": "ECSTaskExecutionRole"
            },
            "ContainerDefinitions": [
                {
                    "Name": {
                        "Ref": "ServiceName"
                    },
                    "Cpu": {
                        "Ref": "ContainerCpu"
                    },
                    "Memory": {
                        "Ref": "ContainerMemory"
                    },
                    "Image": {
                        "Ref": "ImageUrl"
                    },
                    "PortMappings": [
                        {
                            "ContainerPort": {
                                "Ref": "ContainerPort"
                            },
                            "HostPort": {
                                "Ref": "ContainerPort"
                            }
                        }
                    ],
                    "LogConfiguration": {
                        "LogDriver": "awslogs",
                        "Options": {
                            "mode": "non-blocking",
                            "max-buffer-size": "25m",
                            "awslogs-group": {
                                "Ref": "LogGroup"
                            },
                            "awslogs-region": {
                                "Ref": "AWS::Region"
                            },
                            "awslogs-stream-prefix": {
                                "Ref": "ServiceName"
                            }
                        }
```

```
                    }
                }
            ]
        }
    },
    "Service": {
        "Type": "AWS::ECS::Service",
        "DependsOn": "PublicLoadBalancerListener",
        "Properties": {
            "ServiceName": {
                "Ref": "ServiceName"
            },
            "Cluster": {
                "Ref": "ClusterName"
            },
            "PlacementStrategies": [
                {
                    "Field": "attribute:ecs.availability-zone",
                    "Type": "spread"
                },
                {
                    "Field": "cpu",
                    "Type": "binpack"
                }
            ],
            "CapacityProviderStrategy": [
                {
                    "Base": 0,
                    "CapacityProvider": {
                        "Ref": "CapacityProvider"
                    },
                    "Weight": 1
                }
            ],
            "NetworkConfiguration": {
                "AwsvpcConfiguration": {
                    "SecurityGroups": [
                        {
                            "Ref": "ServiceSecurityGroup"
                        }
                    ],
                    "Subnets": {
                        "Ref": "PrivateSubnetIds"
                    }
```

```
                }
            },
            "DeploymentConfiguration": {
                "MaximumPercent": 200,
                "MinimumHealthyPercent": 75
            },
            "DesiredCount": {
                "Ref": "DesiredCount"
            },
            "TaskDefinition": {
                "Ref": "TaskDefinition"
            },
            "LoadBalancers": [
                {
                    "ContainerName": {
                        "Ref": "ServiceName"
                    },
                    "ContainerPort": {
                        "Ref": "ContainerPort"
                    },
                    "TargetGroupArn": {
                        "Ref": "ServiceTargetGroup"
                    }
                }
            ]
        }
    },
    "ServiceSecurityGroup": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupDescription": "Security group for service",
            "VpcId": {
                "Ref": "VpcId"
            }
        }
    },
    "ServiceTargetGroup": {
        "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
        "Properties": {
            "HealthCheckIntervalSeconds": 6,
            "HealthCheckPath": "/",
            "HealthCheckProtocol": "HTTP",
            "HealthCheckTimeoutSeconds": 5,
            "HealthyThresholdCount": 2,
```

```
                "TargetType": "ip",
                "Port": {
                    "Ref": "ContainerPort"
                },
                "Protocol": "HTTP",
                "UnhealthyThresholdCount": 10,
                "VpcId": {
                    "Ref": "VpcId"
                },
                "TargetGroupAttributes": [
                    {
                        "Key": "deregistration_delay.timeout_seconds",
                        "Value": 0
                    }
                ]
            }
        },
        "PublicLoadBalancerSG": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "Access to the public facing load balancer",
                "VpcId": {
                    "Ref": "VpcId"
                },
                "SecurityGroupIngress": [
                    {
                        "CidrIp": "0.0.0.0/0",
                        "IpProtocol": -1
                    }
                ]
            }
        },
        "PublicLoadBalancer": {
            "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
            "Properties": {
                "Scheme": "internet-facing",
                "LoadBalancerAttributes": [
                    {
                        "Key": "idle_timeout.timeout_seconds",
                        "Value": "30"
                    }
                ],
                "Subnets": {
                    "Ref": "PublicSubnetIds"
```

```
            },
            "SecurityGroups": [
                {
                    "Ref": "PublicLoadBalancerSG"
                }
            ]
        }
    },
    "PublicLoadBalancerListener": {
        "Type": "AWS::ElasticLoadBalancingV2::Listener",
        "Properties": {
            "DefaultActions": [
                {
                    "Type": "forward",
                    "ForwardConfig": {
                        "TargetGroups": [
                            {
                                "TargetGroupArn": {
                                    "Ref": "ServiceTargetGroup"
                                },
                                "Weight": 100
                            }
                        ]
                    }
                }
            ],
            "LoadBalancerArn": {
                "Ref": "PublicLoadBalancer"
            },
            "Port": 80,
            "Protocol": "HTTP"
        }
    },
    "ServiceIngressfromLoadBalancer": {
        "Type": "AWS::EC2::SecurityGroupIngress",
        "Properties": {
            "Description": "Ingress from the public ALB",
            "GroupId": {
                "Ref": "ServiceSecurityGroup"
            },
            "IpProtocol": -1,
            "SourceSecurityGroupId": {
                "Ref": "PublicLoadBalancerSG"
            }
```

```
            }
        },
        "LogGroup": {
            "Type": "AWS::Logs::LogGroup"
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: >-
  An example service that deploys in AWS VPC networking mode on EC2 capacity.
  Service uses a capacity provider to request EC2 instances to run on. Service
  runs with networking in private subnets, but still accessible to the internet
  via a load balancer hosted in public subnets.
Parameters:
  VpcId:
    Type: String
    Description: The VPC that the service is running inside of
  PublicSubnetIds:
    Type: 'List<AWS::EC2::Subnet::Id>'
    Description: List of public subnet ID's to put the load balancer in
  PrivateSubnetIds:
    Type: 'List<AWS::EC2::Subnet::Id>'
    Description: List of private subnet ID's that the AWS VPC tasks are in
  ClusterName:
    Type: String
    Description: The name of the ECS cluster into which to launch capacity.
  ECSTaskExecutionRole:
    Type: String
    Description: The role used to start up an ECS task
  CapacityProvider:
    Type: String
    Description: >-
      The cluster capacity provider that the service should use to request
      capacity when it wants to start up a task
  ServiceName:
    Type: String
    Default: web
    Description: A name for the service
  ImageUrl:
    Type: String
```

```
    Default: 'public.ecr.aws/docker/library/nginx:latest'
    Description: >-
      The url of a docker image that contains the application process that will
      handle the traffic for this service
  ContainerCpu:
    Type: Number
    Default: 256
    Description: How much CPU to give the container. 1024 is 1 CPU
  ContainerMemory:
    Type: Number
    Default: 512
    Description: How much memory in megabytes to give the container
  ContainerPort:
    Type: Number
    Default: 80
    Description: What port that the application expects traffic on
  DesiredCount:
    Type: Number
    Default: 2
    Description: How many copies of the service task to run
Resources:
  TaskDefinition:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      Family: !Ref ServiceName
      Cpu: !Ref ContainerCpu
      Memory: !Ref ContainerMemory
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - EC2
      ExecutionRoleArn: !Ref ECSTaskExecutionRole
      ContainerDefinitions:
        - Name: !Ref ServiceName
          Cpu: !Ref ContainerCpu
          Memory: !Ref ContainerMemory
          Image: !Ref ImageUrl
          PortMappings:
            - ContainerPort: !Ref ContainerPort
              HostPort: !Ref ContainerPort
          LogConfiguration:
            LogDriver: awslogs
            Options:
              mode: non-blocking
              max-buffer-size: 25m
```

```
            awslogs-group: !Ref LogGroup
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: !Ref ServiceName
Service:
  Type: AWS::ECS::Service
  DependsOn: PublicLoadBalancerListener
  Properties:
    ServiceName: !Ref ServiceName
    Cluster: !Ref ClusterName
    PlacementStrategies:
      - Field: 'attribute:ecs.availability-zone'
        Type: spread
      - Field: cpu
        Type: binpack
    CapacityProviderStrategy:
      - Base: 0
        CapacityProvider: !Ref CapacityProvider
        Weight: 1
    NetworkConfiguration:
      AwsvpcConfiguration:
        SecurityGroups:
          - !Ref ServiceSecurityGroup
        Subnets: !Ref PrivateSubnetIds
    DeploymentConfiguration:
      MaximumPercent: 200
      MinimumHealthyPercent: 75
    DesiredCount: !Ref DesiredCount
    TaskDefinition: !Ref TaskDefinition
    LoadBalancers:
      - ContainerName: !Ref ServiceName
        ContainerPort: !Ref ContainerPort
        TargetGroupArn: !Ref ServiceTargetGroup
ServiceSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Security group for service
    VpcId: !Ref VpcId
ServiceTargetGroup:
  Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'
  Properties:
    HealthCheckIntervalSeconds: 6
    HealthCheckPath: /
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 5
```

```
      HealthyThresholdCount: 2
      TargetType: ip
      Port: !Ref ContainerPort
      Protocol: HTTP
      UnhealthyThresholdCount: 10
      VpcId: !Ref VpcId
      TargetGroupAttributes:
        - Key: deregistration_delay.timeout_seconds
          Value: 0
  PublicLoadBalancerSG:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - CidrIp: 0.0.0.0/0
          IpProtocol: -1
  PublicLoadBalancer:
    Type: 'AWS::ElasticLoadBalancingV2::LoadBalancer'
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
        - Key: idle_timeout.timeout_seconds
          Value: '30'
      Subnets: !Ref PublicSubnetIds
      SecurityGroups:
        - !Ref PublicLoadBalancerSG
  PublicLoadBalancerListener:
    Type: 'AWS::ElasticLoadBalancingV2::Listener'
    Properties:
      DefaultActions:
        - Type: forward
          ForwardConfig:
            TargetGroups:
              - TargetGroupArn: !Ref ServiceTargetGroup
                Weight: 100
      LoadBalancerArn: !Ref PublicLoadBalancer
      Port: 80
      Protocol: HTTP
  ServiceIngressfromLoadBalancer:
    Type: 'AWS::EC2::SecurityGroupIngress'
    Properties:
      Description: Ingress from the public ALB
      GroupId: !Ref ServiceSecurityGroup
```

```
      IpProtocol: -1
      SourceSecurityGroupId: !Ref PublicLoadBalancerSG
  LogGroup:
    Type: 'AWS::Logs::LogGroup'
```

IAM roles for Amazon ECS

You can use AWS CloudFormation templates to create IAM roles for use with Amazon ECS. For more information about IAM roles for Amazon ECS, see IAM roles for Amazon ECS.

Amazon ECS task execution role

The task execution role grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your behalf. The role is required depending on the requirements of your task. For more information, see Amazon ECS task execution IAM role.

The following template can be used to create a simple task execution role that uses the `AmazonECSTaskExecutionRolePolicy` managed policy.

JSON

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation template for ECS Task Execution Role",
  "Resources": {
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": ["ecs-tasks.amazonaws.com"]
              },
              "Action": ["sts:AssumeRole"],
              "Condition": {
                "ArnLike": {
                  "aws:SourceArn": {
                    "Fn::Sub": "arn:aws:ecs:${AWS::Region}:${AWS::AccountId}:*"
                  }
                },
```

```
                "StringEquals": {
                  "aws:SourceAccount": {
                    "Ref": "AWS::AccountId"
                  }
                }
              }
            }
          ]
        },
        "Path": "/",
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    }
  },
  "Outputs": {
    "ECSTaskExecutionRoleARN": {
      "Description": "ARN of the ECS Task Execution Role",
      "Value": {
        "Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]
      },
      "Export": {
        "Name": {
          "Fn::Sub": "${AWS::StackName}-ECSTaskExecutionRoleARN"
        }
      }
    },
    "ECSTaskExecutionRoleName": {
      "Description": "Name of the ECS Task Execution Role",
      "Value": {
        "Ref": "ECSTaskExecutionRole"
      },
      "Export": {
        "Name": {
          "Fn::Sub": "${AWS::StackName}-ECSTaskExecutionRoleName"
        }
      }
    }
  }
}
```

YAML

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template for ECS Task Execution Role'
Resources:
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: [ecs-tasks.amazonaws.com]
            Action: ['sts:AssumeRole']
            Condition:
              ArnLike:
                aws:SourceArn: !Sub arn:aws:ecs:${AWS::Region}:${AWS::AccountId}:*
              StringEquals:
                aws:SourceAccount: !Ref AWS::AccountId
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
Outputs:
  ECSTaskExecutionRoleARN:
    Description: ARN of the ECS Task Execution Role
    Value: !GetAtt ECSTaskExecutionRole.Arn
    Export:
      Name: !Sub "${AWS::StackName}-ECSTaskExecutionRoleARN"
  ECSTaskExecutionRoleName:
    Description: Name of the ECS Task Execution Role
    Value: !Ref ECSTaskExecutionRole
    Export:
      Name: !Sub "${AWS::StackName}-ECSTaskExecutionRoleName"
```

Amazon ECS best practices

You can use any of the following pages to learn the most important operational best practices for Amazon ECS networking.

Best practice overview	Learn more
Connect applications to the internet	Connect Amazon ECS applications to the internet
Receive inbound connectio ns to Amazon ECS from the internet.	Best practices for receiving inbound connections to Amazon ECS from the internet
Connect Amazon ECS to other AWS services within a VPC	Best practices for connecting Amazon ECS to AWS services from inside your VPC
Network services across AWS accounts and VPCs	Best practices for networkin g Amazon ECS services across AWS accounts and VPCs
Troubleshoot network issues	AWS services for Amazon ECS networking troubleshooting

You can use any of the following pages to learn the most important operational best practices for Fargate on Amazon ECS.

Best practice overview	Learn more
Fargate security	Fargate security best practices in Amazon ECS
Fargate security considera tions	Fargate security considera tions for Amazon ECS

Best practice overview	Learn more
Linux containers on Fargate container image pull behavior	Linux containers on Fargate container image pull behavior for Amazon ECS
Windows containers on Fargate container image pull behavior	Windows containers on Fargate container image pull behavior for Amazon ECS
Fargate task retirement	Task retirement and maintenance for AWS Fargate on Amazon ECS

You can use any of the following pages to learn the most important operational best practices for task definitions.

Best practice overview	Learn more
Container images	Best practices for Amazon ECS container images
Task size	Best practices for Amazon ECS task sizes
Volume best practices	Storage options for Amazon ECS tasks

You can use any of the following pages to learn the most important operational best practices for clusters and capacity.

Best practice overview	Learn more
Fargate security	Fargate security best practices in Amazon ECS

Best practice overview	Learn more	
EC2 container instance security considerations	Amazon EC2 container instance security considera tions for Amazon ECS	
Cluster auto scaling	Optimize Amazon ECS cluster auto scaling	
Operating at scale	Operating Amazon ECS at scale	
Auto scaling and capacity management	Amazon ECS Auto scaling and capacity management best practices	

You can use any of the following pages to learn the most important operational best practices for tasks and services.

Best practice overview	Learn more	
Optimize task launch time	Optimize Amazon ECS task launch time	
Service parameters	Best practices for Amazon ECS service parameters	
Optimize load balancer health check parameters	Optimize load balancer health check parameters for Amazon ECS	
Optimize load balancer connection draining parameters	Optimize load balancer connection draining parameters for Amazon ECS	
Optimize service auto scaling	Optimizing Amazon ECS service auto scaling	

You can use any of the following pages to learn the most important operational best practices for security.

Best practice overview	Learn more	
Network security	Network security best practices for Amazon ECS	
Task and container security	Amazon ECS task and container security best practices	

AWS Fargate for Amazon ECS

AWS Fargate is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task. You configure your task definitions for Fargate by setting the `requiresCompatibilities` task definition parameter to `FARGATE`. For more information, see Launch types.

Fargate offers platform versions for Amazon Linux 2 (platform version 1.3.0), Bottlerocket operating system (platform version 1.4.0), and Microsoft Windows 2019 Server Full and Core editions.Unless otherwise specified, the information on this page applies to all Fargate platforms.

This topic describes the different components of Fargate tasks and services, and calls out special considerations for using Fargate with Amazon ECS.

For information about the Regions that support Linux containers on Fargate, see the section called “Linux containers on AWS Fargate”.

For information about the Regions that support Windows containers on Fargate, see the section called “Windows containers on AWS Fargate”.

Walkthroughs

For information about how to get started using the console, see:

- Learn how to create an Amazon ECS Linux task for the Fargate launch type
- Learn how to create an Amazon ECS Windows task for the Fargate launch type

For information about how to get started using the AWS CLI, see:

- Creating an Amazon ECS Linux task for the Fargate launch type with the AWS CLI
- Creating an Amazon ECS Windows task for the Fargate launch type with the AWS CLI

Capacity providers

The following capacity providers are available:

- Fargate
- Fargate Spot - Run interruption tolerant Amazon ECS tasks at a discounted rate compared to the AWS Fargate price. Fargate Spot runs tasks on spare compute capacity. When AWS needs the capacity back, your tasks will be interrupted with a two-minute warning. For more information, see Amazon ECS clusters for Fargate.

Task definitions

Tasks that use the Fargate launch type don't support all of the Amazon ECS task definition parameters that are available. Some parameters aren't supported at all, and others behave differently for Fargate tasks. For more information, see Task CPU and memory.

Platform versions

AWS Fargate platform versions are used to refer to a specific runtime environment for Fargate task infrastructure. It is a combination of the kernel and container runtime versions. You select a platform version when you run a task or when you create a service to maintain a number of identical tasks.

New revisions of platform versions are released as the runtime environment evolves, for example, if there are kernel or operating system updates, new features, bug fixes, or security updates. A Fargate platform version is updated by making a new platform version revision. Each task runs on one platform version revision during its lifecycle. If you want to use the latest platform version revision, then you must start a new task. A new task that runs on Fargate always runs on the latest revision of a platform version, ensuring that tasks are always started on secure and patched infrastructure.

If a security issue is found that affects an existing platform version, AWS creates a new patched revision of the platform version and retires tasks running on the vulnerable revision. In some cases, you may be notified that your tasks on Fargate have been scheduled for retirement. For more information, see Task retirement and maintenance for AWS Fargate on Amazon ECS .

For more information see Fargate platform versions for Amazon ECS.

Service load balancing

Your Amazon ECS service on AWS Fargate can optionally be configured to use Elastic Load Balancing to distribute traffic evenly across the tasks in your service.

Amazon ECS services on AWS Fargate support the Application Load Balancer, Network Load Balancer, and Gateway Load Balancer load balancer types. Application Load Balancers are used to route HTTP/HTTPS (or layer 7) traffic. Network Load Balancers are used to route TCP or UDP (or layer 4) traffic. For more information, see Use load balancing to distribute Amazon ECS service traffic.

When you create a target group for these services, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance. For more information, see Use load balancing to distribute Amazon ECS service traffic.

Using a Network Load Balancer to route UDP traffic to your Amazon ECS on AWS Fargate tasks is only supported when using platform version 1.4 or later.

Usage metrics

You can use CloudWatch usage metrics to provide visibility into your accounts usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

AWS Fargate usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about AWS Fargate service quotas, Amazon ECS endpoints and quotas in the *Amazon Web Services General Reference*..

For more information about AWS Fargate usage metrics, see AWS Fargate usage metrics.

Amazon ECS security considerations for when to use the Fargate launch type

We recommend that customers looking for strong isolation for their tasks use Fargate. Fargate runs each task in a hardware virtualization environment. This ensures that these containerized workloads do not share network interfaces, Fargate ephemeral storage, CPU, or memory with other tasks. For more information, see Security Overview of AWS Fargate.

Fargate security best practices in Amazon ECS

We recommend that you take into account the following best practices when you use AWS Fargate. For additional guidance, see Security overview of AWS Fargate.

Use AWS KMS to encrypt ephemeral storage for Fargate

You should have your ephemeral storage encrypted by either AWS KMS or your own customer managed keys. For tasks that are hosted on Fargate using platform version `1.4.0` or later, each task receives 20 GiB of ephemeral storage. For more information, see customer managed key (CMK). You can increase the total amount of ephemeral storage, up to a maximum of 200 GiB, by specifying the `ephemeralStorage` parameter in your task definition. For such tasks that were launched on May 28, 2020 or later, the ephemeral storage is encrypted with an AES-256 encryption algorithm using an encryption key managed by Fargate.

For more information, see Storage options for Amazon ECS tasks.

Example: Launching an task on Fargate platform version 1.4.0 with ephemeral storage encryption

The following command will launch a task on Fargate platform version 1.4. Because this task is launched as part of the cluster, it uses the 20 GiB of ephemeral storage that's automatically encrypted.

```
aws ecs run-task --cluster clustername \
  --task-definition taskdefinition:version \
  --count 1
  --launch-type "FARGATE" \
  --platform-version 1.4.0 \
  --network-configuration
 "awsvpcConfiguration={subnets=[subnetid],securityGroups=[securitygroupid]}" \
  --region region
```

SYS_PTRACE capability for kernel syscall tracing with Fargate

The default configuration of Linux capabilities that are added or removed from your container are provided by Docker.

Tasks that are launched on Fargate only support adding the SYS_PTRACE kernel capability.

The following video shows how to use this feature through the Sysdig Falco project.

#ContainersFromTheCouch - Troubleshooting your Fargate Task using SYS_PTRACE capability

The code discussed in the previous video can be found on GitHub here.

Use Amazon GuardDuty with Fargate Runtime Monitoring

Amazon GuardDuty is a threat detection service that helps protect your accounts, containers, workloads, and the data within your AWS environment. Using machine learning (ML) models, and anomaly and threat detection capabilities, GuardDuty continuously monitors different log sources and runtime activity to identify and prioritize potential security risks and malicious activities in your environment.

Runtime Monitoring in GuardDuty protects workloads running on Fargate by continuously monitoring AWS log and networking activity to identify malicious or unauthorized behavior. Runtime Monitoring uses a lightweight, fully managed GuardDuty security agent that analyzes on-host behavior, such as file access, process execution, and network connections. This covers issues including escalation of privileges, use of exposed credentials, or communication with malicious IP addresses, domains, and the presence of malware on your Amazon EC2 instances and container workloads. For more information, see GuardDuty Runtime Monitoring in the *GuardDuty User Guide*.

Fargate security considerations for Amazon ECS

Each task has a dedicated infrastructure capacity because Fargate runs each workload on an isolated virtual environment. Workloads that run on Fargate do not share network interfaces, ephemeral storage, CPU, or memory with other tasks. You can run multiple containers within a task including application containers and sidecar containers, or simply sidecars. A *sidecar* is a container that runs alongside an application container in an Amazon ECS task. While the application container runs core application code, processes running in sidecars can augment the application. Sidecars help you segregate application functions into dedicated containers, making it easier for you to update parts of your application.

Containers that are part of the same task share resources for the Fargate launch type because these containers will always run on the same host and share compute resources. These containers also share the ephemeral storage provided by Fargate. Linux containers in a task share network namespaces, including the IP address and network ports. Inside a task, containers that belong to the task can inter-communicate over localhost.

The runtime environment in Fargate prevents you from using certain controller features that are supported on EC2 instances. Consider the following when you architect workloads that run on Fargate:

- No privileged containers or access - Features such as privileged containers or access are currently unavailable on Fargate. This will affect uses cases such as running Docker in Docker.
- Limited access to Linux capabilities - The environment in which containers run on Fargate is locked down. Additional Linux capabilities, such as CAP_SYS_ADMIN and CAP_NET_ADMIN, are restricted to prevent a privilege escalation. Fargate supports adding the CAP_SYS_PTRACE Linux capability to tasks to allow observability and security tools deployed within the task to monitor the containerized application.
- No access to the underlying host - Neither customers nor AWS operators can connect to a host running customer workloads. You can use ECS exec to run commands in or get a shell to a container running on Fargate. You can use ECS exec to help collect diagnostic information for debugging. Fargate also prevents containers from accessing the underlying host's resources, such as the file system, devices, networking, and container runtime.
- Networking - You can use security groups and network ACLs to control inbound and outbound traffic. Fargate tasks receive an IP address from the configured subnet in your VPC.

Fargate platform versions for Amazon ECS

AWS Fargate platform versions are used to refer to a specific runtime environment for Fargate task infrastructure. It is a combination of the kernel and container runtime versions. You select a platform version when you run a task or when you create a service to maintain a number of identical tasks.

New revisions of platform versions are released as the runtime environment evolves, for example, if there are kernel or operating system updates, new features, bug fixes, or security updates. A Fargate platform version is updated by making a new platform version revision. Each task runs on one platform version revision during its lifecycle. If you want to use the latest platform version revision, then you must start a new task. A new task that runs on Fargate always runs on the latest revision of a platform version, ensuring that tasks are always started on secure and patched infrastructure.

If a security issue is found that affects an existing platform version, AWS creates a new patched revision of the platform version and retires tasks running on the vulnerable revision. In some cases, you may be notified that your tasks on Fargate have been scheduled for retirement. For more information, see Task retirement and maintenance for AWS Fargate on Amazon ECS .

You specify the platform version when you run a task, or deploy a service.

Consider the following when specifying a platform version:

- You can specify a a specific version number, for example `1.4.0`, or `LATEST`.

  The **LATEST** Linux platform version is `1.4.0`.

  The **LATEST** Windows platform version is `1.0.0`.
- If you want to update the platform version for a service, create a deployment. For example, assume that you have a service that runs tasks on the Linux platform version `1.3.0`. To change the service to run tasks on the Linux platform version `1.4.0`, you update your service and specify a new platform version. Your tasks are redeployed with the latest platform version and the latest platform version revision. For more information about deployments, see Amazon ECS services.
- If your service is scaled up without updating the platform version, those tasks receive the platform version that was specified on the service's current deployment. For example, assume that you have a service that runs tasks on the Linux platform version `1.3.0`. If you increase the desired count of the service, the service scheduler starts the new tasks using the latest platform version revision of platform version `1.3.0`.
- New tasks always run on the latest revision of a platform version. This ensures tasks are always on secured and patched infrastructure.
- The platform version numbers for Linux containers and Windows containers on Fargate are independent. For example, the behavior, features, and software used in platform version `1.0.0` for Windows containers on Fargate aren't comparable to those of platform version `1.0.0` for Linux containers on Fargate.
- The following applies to Fargate Windows platform versions.

  Microsoft Windows Server container images must be created from a specific version of Windows Server. You must select the same version of Windows Server in the `platformFamily` when you run a task or create a service that matches the Windows Server container image. Additionally, you can provide a matching `operatingSystemFamily` in the task definition to prevent tasks from being run on the wrong Windows version. For more information, see Matching container host version with container image versions on the Microsoft Learn website.

Migrating to Linux platform version 1.4.0 on Amazon ECS

Consider the following when migrating your Amazon ECS on Fargate tasks from platform version `1.0.0`, `1.1.0`, `1.2.0`, or `1.3.0` to platform version `1.4.0`. It is best practice to confirm your task works properly on platform version `1.4.0` before you migrate the tasks.

- The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4.0, all Amazon ECS on Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC. The traffic is visible to you through your VPC flow logs. For more information see Amazon ECS task networking options for the Fargate launch type.
- If you use interface VPC endpoints, consider the following.
  - For container images hosted with Amazon ECR, you need the following endpoints. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.
    - **com.amazonaws.*region*.ecr.dkr** Amazon ECR VPC endpoint
    - **com.amazonaws.*region*.ecr.api** Amazon ECR VPC endpoint
    - Amazon S3 gateway endpoint
  - When your task definition references Secrets Manager secrets to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Secrets Manager. For more information, see Using Secrets Manager with VPC Endpoints in the *AWS Secrets Manager User Guide*.
  - When your task definition references Systems Manager Parameter Store parameters to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Systems Manager. For more information, see Improve the security of EC2 instances by using VPC endpoints for Systems Manager in the *AWS Systems Manager User Guide*.
  - The security group for the Elastic Network Interface (ENI) associated with your task needs the security group rules to allow traffic between the task and the VPC endpoints.

Fargate Linux platform version change log

The following are the available Linux platform versions. For information about platform version deprecation, see AWS Fargate Linux platform version deprecation.

1.4.0

The following is the changelog for platform version `1.4.0`.

- Beginning on November 5, 2020, any new Amazon ECS task launched on Fargate using platform version `1.4.0` will be able to use the following features:
  - When using Secrets Manager to store sensitive data, you can inject a specific JSON key or a specific version of a secret as an environment variable or in a log configuration. For more information, see Pass sensitive data to an Amazon ECS container.
  - Specify environment variables in bulk using the `environmentFiles` container definition parameter. For more information, see Pass an individual environment variable to an Amazon ECS container.
  - Tasks run in a VPC and subnet enabled for IPv6 will be assigned both a private IPv4 address and an IPv6 address. For more information, see Amazon ECS task networking options for the Fargate launch type.
  - The task metadata endpoint version 4 provides additional metadata about your task and container including the task launch type, the Amazon Resource Name (ARN) of the container, and the log driver and log driver options used. When querying the `/stats` endpoint you also receive network rate stats for your containers. For more information, see Task metadata endpoint version 4.
- Beginning on July 30, 2020, any new Amazon ECS task launched on Fargate using platform version `1.4.0` will be able to route UDP traffic using a Network Load Balancer to their Amazon ECS on Fargate tasks. For more information, see Use load balancing to distribute Amazon ECS service traffic.
- Beginning on May 28, 2020, any new Amazon ECS task launched on Fargate using platform version `1.4.0` will have its ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS owned encryption key. For more information, see Fargate task ephemeral storage for Amazon ECS and Storage options for Amazon ECS tasks.
- Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see Use Amazon EFS volumes with Amazon ECS.
- The ephemeral task storage has been increased to a minimum of 20 GB for each task. For more information, see Fargate task ephemeral storage for Amazon ECS.
- The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4.0, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through

your VPC flow logs. For more information about networking for the Amazon EC2 launch type, see Amazon ECS task networking options for the EC2 launch type. For more information about networking for the Fargate launch type, see Amazon ECS task networking options for the Fargate launch type.

- Task ENIs add support for jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames will reduce overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC.
- CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see Monitor Amazon ECS containers using Container Insights with enhanced observability.
- Added support for the task metadata endpoint version 4 which provides additional information for your Fargate tasks, including network stats for the task and which Availability Zone the task is running in. For more information, see Amazon ECS task metadata endpoint version 4 and Amazon ECS task metadata endpoint version 4 for tasks on Fargate.
- Added support for the `SYS_PTRACE` Linux parameter in container definitions. For more information, see Linux parameters.
- The Fargate container agent replaces the use of the Amazon ECS container agent for all Fargate tasks. Usually, this change does not have an effect on how your tasks run.
- The container runtime is now using Containerd instead of Docker. Most likely, this change does not have an effect on how your tasks run. You will notice that some error messages that originate with the container runtime changes from mentioning Docker to more general errors. For more information, see Amazon ECS stopped task error messages.
- Based on Amazon Linux 2.

AWS Fargate Linux platform version deprecation

Important

On March 2, 2026, AWS will start force-updating all the Amazon ECS services running Fargate platform version 1.3.0 that are not migrated to the platform version 1.4.0. That may potentially impact your workloads. Starting March 16, 2026, we will make the platform version 1.3.0 as Retired. At that time, you will not be able to launch new tasks

or create new services configured with platform version 1.3.0, but your existing tasks will continue running. On March 31, 2026, we will terminate all the remaining running tasks configured with platform version 1.3.0.
For information about how to migrate to platform version 1.4, see [Migrating to Linux platform version 1.4.0 on Amazon ECS](#).

The following table lists Linux platform versions that AWS Fargate has deprecated or have been scheduled for deprecation. These platform versions remain available until the published deprecation date.

A *force update date* is provided for each platform version scheduled for deprecation. On the force update date, any service using the `LATEST` platform version that is pointed to a platform version that is scheduled for deprecation will be updated using the force new deployment option. When the service is updated using the force new deployment option, all tasks running on a platform version scheduled for deprecation are stopped and new tasks are launched using the platform version that the `LATEST` tag points to at that time. Standalone tasks or services with an explicit platform version set are not affected by the force update date.

For information about how to migrate to latest platform version, see [Migrating to Linux platform version 1.4.0 on Amazon ECS](#).

After a platform version reaches the *deprecation date*, the platform version will no longer be available for new tasks or services. Any standalone tasks or services which explicitly use a deprecated platform version will continue using that platform version until the tasks are stopped. After the deprecation date, a deprecated platform version will no longer receive any security updates or bug fixes.

Platform version	Force update date	Deprecation date
1.0.0	October 26, 2020	December 14, 2020
1.1.0	October 26, 2020	December 14, 2020
1.2.0	October 26, 2020	December 14, 2020
1.3.0	March 2, 2026	March 16, 2026

For information about current platform versions, see Fargate platform versions for Amazon ECS.

Deprecated Fargate Linux versions change log

1.3.0

The following is the changelog for platform version `1.3.0`.

- Beginning on Sept 30, 2019, any new Fargate task that is launched supports the `awsfirelens` log driver. Configure the FireLens for Amazon ECS to use task definition parameters to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.
- Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service. For more information, Task retirement and maintenance for AWS Fargate on Amazon ECS.
- Beginning on March 27, 2019, any new Fargate task that is launched can use additional task definition parameters that you use to define a proxy configuration, dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Proxy configuration, Container dependency, and Container timeouts.
- Beginning on April 2, 2019, any new Fargate task that is launched supports injecting sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. For more information, see Pass sensitive data to an Amazon ECS container.
- Beginning on May 1, 2019, any new Fargate task that is launched supports referencing sensitive data in the log configuration of a container using the `secretOptions` container definition parameter. For more information, see Pass sensitive data to an Amazon ECS container.
- Beginning on May 1, 2019, any new Fargate task that is launched supports the `splunk` log driver in addition to the `awslogs` log driver. For more information, see Storage and logging.
- Beginning on July 9, 2019, any new Fargate tasks that is launched supports CloudWatch Container Insights. For more information, see Monitor Amazon ECS containers using Container Insights with enhanced observability.
- Beginning on December 3, 2019, the Fargate Spot capacity provider is supported. For more information, see Amazon ECS clusters for Fargate.
- Based on Amazon Linux 2.

1.2.0

The following is the changelog for platform version `1.2.0`.

> **Note**
>
> Platform version `1.2.0` is no longer available. For information about platform version deprecation, see AWS Fargate Linux platform version deprecation.

- Added support for private registry authentication using AWS Secrets Manager. For more information, see Using non-AWS container images in Amazon ECS.

1.1.0

The following is the changelog for platform version `1.1.0`.

> **Note**
>
> Platform version `1.1.0` is no longer available. For information about platform version deprecation, see AWS Fargate Linux platform version deprecation.

- Added support for the Amazon ECS task metadata endpoint. For more information, see Amazon ECS task metadata available for tasks on Fargate.
- Added support for Docker health checks in container definitions. For more information, see Health check.
- Added support for Amazon ECS service discovery. For more information, see Use service discovery to connect Amazon ECS services with DNS names.

1.0.0

The following is the changelog for platform version `1.0.0`.

> **Note**
>
> Platform version `1.0.0` is no longer available. For information about platform version deprecation, see AWS Fargate Linux platform version deprecation.

- Based on Amazon Linux 2017.09.
- Initial release.

Fargate Windows platform version change log

The following are the available platform versions for Windows containers.

1.0.0

The following is the changelog for platform version `1.0.0`.

- Initial release for support on the following Microsoft Windows Server operating systems:
  - Windows Server 2019 Full
  - Windows Server 2019 Core
  - Windows Server 2022 Full
  - Windows Server 2022 Core

Windows containers on Fargate considerations for Amazon ECS

The following are the differences and considerations to know when you run Windows containers on AWS Fargate.

If you need to run tasks on Linux and Windows containers, then you need to create separate task definitions for each operating system.

AWS handles the operating system license management, so you do not need any additional Microsoft Windows Server licenses.

Windows containers on AWS Fargate supports the following operating systems:

- Windows Server 2019 Full
- Windows Server 2019 Core
- Windows Server 2022 Full
- Windows Server 2022 Core

Windows containers on AWS Fargate supports the awslogs driver. For more information, see the section called “Send logs to CloudWatch ”.

The following features are not supported on Windows containers on Fargate:

- Amazon FSx
- ENI trunking
- gMSAs for Windows Containers
- App Mesh service and proxy integration for tasks
- Firelens log router integration for tasks
- EFS volumes
- EBS volumes
- The following task definition parameters:
  - `maxSwap`
  - `swappiness`
  - `environmentFiles`
- The Fargate Spot capacity provider
- Image volumes

  The Dockerfile `volume` option is ignored. Instead, use bind mounts in your task definition. For more information, see [Use bind mounts with Amazon ECS](#).
- The task-level CPU and memory parameters are ignored for Windows containers. We recommend specifying container-level resources for Windows containers.
- memory for task
- mermoryReservation on containers
- Restart policies on containers

Fargate task ephemeral storage for Amazon ECS

When provisioned, each Amazon ECS task hosted on Linux containers on AWS Fargate receives the following ephemeral storage for bind mounts. This can be mounted and shared among containers that use the `volumes`, `mountPoints`, and `volumesFrom` parameters in the task definition. This isn't supported for Windows containers on AWS Fargate.

Fargate Linux container platform versions

Version 1.4.0 or later

By default, Amazon ECS tasks that are hosted on Fargate using platform version `1.4.0` or later receive a minimum of 20 GiB of ephemeral storage. The total amount of ephemeral storage can be increased, up to a maximum of 200 GiB. You can do this by specifying the `ephemeralStorage` parameter in your task definition.

The pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage your task has to use, you must subtract the amount of storage your container image uses from the total amount of ephemeral storage your task is allocated.

For tasks that use platform version `1.4.0` or later that are launched on May 28, 2020 or later, the ephemeral storage is encrypted with an AES-256 encryption algorithm. This algorithm uses an AWS owned encryption key, or you can create your own customer managed key. For more information, see Customer managed keys for AWS Fargate ephemeral storage.

For tasks that use platform version `1.4.0` or later that are launched on November 18, 2022 or later, the ephemeral storage usage is reported through the task metadata endpoint. Your applications in your tasks can query the task metadata endpoint version 4 to get their ephemeral storage reserved size and the amount used.

Additionally, the ephemeral storage reserved size and the amount used are sent to Amazon CloudWatch Container Insights if you turn on Container Insights.

> **Note**
>
> Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as `df`.

Version 1.3.0 or earlier

For Amazon ECS on Fargate tasks that use platform version `1.3.0` or earlier, each task receives the following ephemeral storage.

- 10 GB of Docker layer storage

> **Note**
>
> This amount includes both compressed and uncompressed container image artifacts.

- An additional 4 GB for volume mounts. This can be mounted and shared among containers that use the `volumes`, `mountPoints`, and `volumesFrom` parameters in the task definition.

Fargate Windows container platform versions

Version 1.0.0 or later

By default, Amazon ECS tasks that are hosted on Fargate using platform version `1.0.0` or later receive a minimum of 20 GiB of ephemeral storage. The total amount of ephemeral storage can be increased, up to a maximum of 200 GiB. You can do this by specifying the `ephemeralStorage` parameter in your task definition.

The pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage that your task has to use, you must subtract the amount of storage that your container image uses from the total amount of ephemeral storage your task is allocated.

For more information, see Use bind mounts with Amazon ECS.

Customer managed keys for AWS Fargate ephemeral storage for Amazon ECS

AWS Fargate supports customer managed keys to encrypt data for Amazon ECS tasks stored in ephemeral storage to help regulation-sensitive customers meet their internal security policies. Customers still get the serverless benefit of Fargate, while giving enhanced visibility on self-managed storage encryption to compliance auditors. While Fargate has Fargate-managed ephemeral storage encryption by default, customers can also use their own self-managed keys when encrypting sensitive data like financial or health related information.

You can import your own keys into AWS KMS or create the keys in AWS KMS. These self-managed keys are stored in AWS KMS and perform standard AWS KMS lifecycle actions such as rotate, disable, and delete. You can audit key access and usage in CloudTrail logs.

By default, KMS key supports 50,000 grants per key. Fargate uses a single AWS KMS grant per customer managed key task, so it supports up to 50,000 concurrent tasks for a key. If you want to increase this number, you can ask for a limit increase, which is approved on a case-by-case basis.

Fargate doesn't charge anything extra for using customer managed keys. You're only charged the standard price for using AWS KMS keys for storage and API requests.

Topics

- Create an encryption key for Fargate ephemeral storage for Amazon ECS
- Managing AWS KMS keys for Fargate ephemeral storage for Amazon ECS

Create an encryption key for Fargate ephemeral storage for Amazon ECS

Create a customer managed key to encrypt data stored on Fargate ephemeral storage.

> **Note**
>
> Fargate ephemeral storage encryption with customer managed keys isn't available for Windows task clusters.
> Fargate ephemeral storage encryption with customer managed keys isn't available on `platformVersions` earlier than `1.4.0`.
> Fargate reserves space on an ephemeral storage that's only used by Fargate, and you're not billed for the space. Allocation might differ from non-customer managed key tasks, but the total space remains the same. You can view this change in tools like `df`.
> Multi-Region keys are not supported for Fargate ephemeral storage.
> KMS key aliases are not supported for Fargate ephemeral storage.

To create a customer managed key (CMK) to encrypt ephemeral storage for Fargate in AWS KMS, follow these steps.

1. Navigate to the https://console.aws.amazon.com/kms.
2. Follow the instructions for Creating Keys in the AWS Key Management Service Developer Guide.
3. When creating your AWS KMS key, make sure to provide Fargate service relevant AWS KMS operation permissions in the key policies. The following API operations must be permitted in the policy to use your customer managed key with your Amazon ECS cluster resources.

- `kms:GenerateDataKeyWithoutPlainText` - Call `GenerateDataKeyWithoutPlainText` to generate an encrypted data key from the provided AWS KMS key.
- `kms:CreateGrant` - Adds a grant to a customer managed key. Grants control access to a specified AWS KMS key, which allows access to grant operations that Amazon ECS Fargate requires. For more information about [Using Grants](), see the [AWS Key Management Service Developer Guide](). This allows Amazon ECS Fargate to do the following:
  - Call `Decrypt` to AWS KMS to get the encryption key to decrypt the ephemeral storage data.
  - Set up a retiring principal to allow the service to `RetireGrant`.
- `kms:DescribeKey` - Provides the customer managed key details to allow Amazon ECS to validate the key if it's symmetric and enabled.

The following example shows a AWS KMS key policy that you would apply to the target key for encryption. To use the example policy statements, replace the *`user input placeholders`* with your own information. As always, only configure the permissions that you need, but you'll need to provide AWS KMS with permissions to at least one user to avoid errors.

```
{
      "Sid": "Allow generate data key access for Fargate tasks.",
      "Effect": "Allow",
      "Principal": { "Service":"fargate.amazonaws.com" },
      "Action": [
        "kms:GenerateDataKeyWithoutPlaintext"
      ],
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:ecs:clusterAccount": [
            "customerAccountId"
          ],
          "kms:EncryptionContext:aws:ecs:clusterName": [
             "clusterName"
          ]
        }
      },
      "Resource": "*"
    },
    {
```

```
      "Sid": "Allow grant creation permission for Fargate tasks.",
      "Effect": "Allow",
      "Principal": { "Service":"fargate.amazonaws.com" },
      "Action": [
        "kms:CreateGrant"
      ],
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:ecs:clusterAccount": [
            "customerAccountId"
          ],
          "kms:EncryptionContext:aws:ecs:clusterName": [
             "clusterName"
          ]
        },
       "ForAllValues:StringEquals": {
          "kms:GrantOperations": [
             "Decrypt"
          ]
       }
      },
      "Resource": "*"
    },
    {
      "Sid": "Allow describe key permission for cluster operator - CreateCluster
 and UpdateCluster.",
      "Effect": "Allow",
      "Principal": { "AWS":"arn:aws:iam::customerAccountId:role/customer-chosen-
role" },
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
```

Fargate tasks use the `aws:ecs:clusterAccount` and `aws:ecs:clusterName` encryption context keys for cryptographic operations with the key. Customers should add these permissions to restrict access to a specific account and/or cluster. Use the cluster name and not the ARN when you specify the cluster.

For more information, see Encryption context in the AWS KMS Developer Guide.

When creating or updating a cluster, you have the option to use the condition key `fargateEphemeralStorageKmsKeyId`. This condition key allows customers to have more granular control of the IAM policies. Updates to the `fargateEphemeralStorageKmsKeyId` configuration only take effect on new service deployments.

The following is an example of allowing customers to grant permissions to only a specific set of approved AWS KMS keys.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:UpdateCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "ecs:fargate-ephemeral-storage-kms-key": "arn:aws:kms:us-
west-2:111122223333:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        }
      }
    }
  ]
}
```

Next is an example for denying attempts to remove AWS KMS keys that are already associated with a cluster.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
```

```
        "Effect": "Deny",
        "Action": [
            "ecs:CreateCluster",
            "ecs:UpdateCluster"
        ],
        "Resource": "*",
        "Condition": {
          "Null": {
            "ecs:fargate-ephemeral-storage-kms-key": "true"
          }
        }
      }
    }
```

Customers can see if their unmanaged tasks or service tasks are encrypted using the key by using the AWS CLI `describe-tasks`, `describe-cluster`, or `describe-services` commands.

For more information, see Condition keys for AWS KMS in the AWS KMS Developer Guide.

AWS Management Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. Choose **Clusters** in the left navigation and either **Create cluster** in the top right, or choose an existing cluster. For an existing cluster, choose **Update cluster** in the top right.
3. Under the **Encryption** section of the workflow, you'll have the option to select your AWS KMS key under **Managed storage** and **Fargate ephemeral storage**. You can also choose to **Create an AWS KMS key** from here.
4. Choose **Create** once you finish creating your new cluster or **Update**, if you were updating an existing one.

AWS CLI

The following is an example of creating a cluster and configuring your Fargate ephemeral storage using the AWS CLI (replace the *red* values with your own):

```
aws ecs create-cluster --cluster clusterName \
```

```
--configuration '{"managedStorageConfiguration":
{"fargateEphemeralStorageKmsKeyId":"arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"}}'
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:012345678901:cluster/clusterName",
        "clusterName": "clusterName",
        "configuration": {
            "managedStorageConfiguration": {
                "fargateEphemeralStorageKmsKeyId": "arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
            }
        },
        "status": "ACTIVE",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [],
        "capacityProviders": [],
        "defaultCapacityProviderStrategy": []
    },
    "clusterCount": 5
}
```

AWS CloudFormation

The following is an example template of creating a cluster and configuring your Fargate ephemeral storage using the AWS CloudFormation (replace the *red* values with your own):

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  MyCluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: "clusterName"
      Configuration:
        ManagedStorageConfiguration:
          FargateEphemeralStorageKmsKeyId: "arn:aws:kms:us-
west-2:012345678901:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Managing AWS KMS keys for Fargate ephemeral storage for Amazon ECS

After creating or importing your AWS KMS key to encrypt your Fargate ephemeral storage, you manage it the same way you would any other AWS KMS key.

Automatic rotation of AWS KMS keys

You can enable automatic key rotation or rotate them manually. Automatic key rotation rotates the key for you yearly by generating new cryptographic material for the key. AWS KMS also saves all previous versions of the cryptographic material, so you'll be able to decrypt any data that used the earlier key versions. Any rotated material won't be deleted by AWS KMS until you delete the key.

Automatic key rotation is optional and can be enabled or disabled at any time.

Disabling or revoking AWS KMS keys

If you disable a customer managed key in AWS KMS, it doesn't have any impact on running tasks, and they continue to function through their lifecycle. If a new task uses the disabled or revoked key, the task fails since it can't access the key. You should set a CloudWatch alarm or similar to make sure a disabled key is never needed to decrypt already encrypted data.

Deleting AWS KMS keys

Deleting keys should always be a last resort and should only be done if you're certain the deleted key is never needed again. New tasks that try to use the deleted key will fail because they can't access it. AWS KMS advises disabling a key instead of deleting it. If you feel it's necessary to delete a key, we suggest disabling it first and setting a CloudWatch alarm to make sure it isn't needed. If you do delete a key, AWS KMS supplies at least seven days to change your mind.

Auditing AWS KMS key access

You can use CloudTrail logs to audit access to your AWS KMS key. You're able to check the AWS KMS operations `CreateGrant`, `GenerateDataKeyWithoutPlaintext`, and `Decrypt`. These operations also show the `aws:ecs:clusterAccount` and `aws:ecs:clusterName` as part of the `EncryptionContext` logged in CloudTrail.

The following are example CloudTrail events for `GenerateDataKeyWithoutPlaintext`, `GenerateDataKeyWithoutPlaintext (DryRun)`, `CreateGrant`, `CreateGrant (DryRun)`, and `RetireGrant` (replace the *red* values with your own).

GenerateDataKeyWithoutPlaintext


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ec2-frontend-api.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:13Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ec2-frontend-api.amazonaws.com",
    "userAgent": "ec2-frontend-api.amazonaws.com",
    "requestParameters": {
        "numberOfBytes": 64,
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "encryptionContext": {
            "aws:ecs:clusterAccount": "account-id",
            "aws:ebs:id": "vol-xxxxxxx",
            "aws:ecs:clusterName": "cluster-name"
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

GenerateDataKeyWithoutPlaintext (DryRun)


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "fargate.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "fargate.amazonaws.com",
    "userAgent": "fargate.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "dryRun": true,
        "numberOfBytes": 64,
        "encryptionContext": {
            "aws:ecs:clusterAccount": "account-id",
            "aws:ecs:clusterName": "cluster-name"
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
```

```
}
```

CreateGrant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "ec2-frontend-api.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:13Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ec2-frontend-api.amazonaws.com",
    "userAgent": "ec2-frontend-api.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "granteePrincipal": "fargate.us-west-2.amazonaws.com",
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecs:clusterAccount": "account-id",
                "aws:ebs:id": "vol-xxxx",
                "aws:ecs:clusterName": "cluster-name"
            }
        },
        "retiringPrincipal": "ec2.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
```

```
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

CreateGrant (DryRun)

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "fargate.amazonaws.com"
    },
    "eventTime": "2024-04-23T18:08:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "fargate.amazonaws.com",
    "userAgent": "fargate.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "granteePrincipal": "fargate.us-west-2.amazonaws.com",
        "dryRun": true,
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:ecs:clusterAccount": "account-id",
                "aws:ecs:clusterName": "cluster-name"
            }
```

```
        }
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

RetireGrant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2024-04-20T18:37:38Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "RetireGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": null,
    "responseElements": {
        "keyId": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
    },
    "additionalEventData": {
        "grantId":
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
```

```
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-west-2:account-id:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-id",
    "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventCategory": "Management"
}
```

Task retirement and maintenance for AWS Fargate on Amazon ECS

AWS is responsible for maintaining the underlying infrastructure for AWS Fargate. AWS determines when a platform version revision needs to be replaced with a new revision for the infrastructure. This is known as task retirement. AWS sends a task retirement notification when a platform version revision is retired. We routinely update our supported platform versions to introduce a new revision containing updates to the Fargate runtime software and underlying dependencies such as the operating system and container runtime. After a newer revision is made available, we retire the older revision in order to ensure all customer workloads run on the most up to date revision of the Fargate platform version. When a revision is retired, all tasks running on that revision are stopped.

Amazon ECS tasks can be categorized as either service tasks or standalone tasks. Service tasks are deployed as part of a service and controlled by the Amazon ECS schedule. For more information, see Amazon ECS services. Standalone tasks are tasks started by the Amazon ECS `RunTask` API, either directly or by an external scheduler such as scheduled tasks (which are started by Amazon EventBridge), AWS Batch, or AWS Step Functions. You do not need to take any actions in response to task retirement for your service tasks because the Amazon ECS scheduler automatically replaces the tasks.

For standalone tasks, you may need to perform additional handling in response to task retirement. For more information, see [Can Amazon ECS automatically handle standalone tasks?](#).

For service tasks, you do not need to take any action to task retirement unless you want to replace these tasks before AWS does. When the Amazon ECS scheduler stops the tasks, it uses the `maximumPercent` and launches a new task in an attempt to maintain the desired count for the service. Follow best practices to minimize the impact of task retirement. The default `maximumPercent` value for a service using the REPLICA service scheduler is 200%. Therefore, when AWS Fargate starts retiring tasks, Amazon ECS first schedules a new task, and then waits for it to be running, before retiring an old task. When you set the `maximumPercent` value to 100%, Amazon ECS stops the task first, then replaces it.

For standalone task retirement, AWS stops the task on or after the task retirement date. Amazon ECS doesn't launch a replacement task when a task is stopped. If you need these tasks to continue to run, you need to stop the running tasks and launch a replacement task before the time indicated in the notification. Therefore, we recommend that customers monitor the state of standalone tasks and if required, implement logic to replace the stopped tasks.

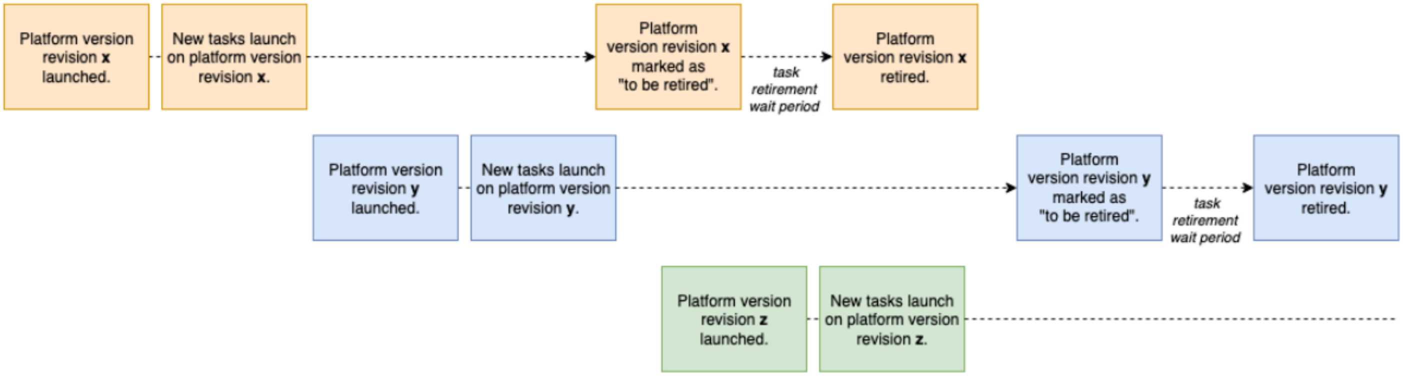
When a task is stopped in any of the scenarios, you can run `describe-tasks`. The `stoppedReason` in the response is `ECS is performing maintenance on the underlying infrastructure hosting the task`.

Task maintenance applies when there is a new platform version revision needs to be replaced with a new revision. If there is an issue with an underlying Fargate host, Amazon ECS replaces the host without a task retirement notice.

Task retirement notice overview

When AWS marks a platform version revision as needing to be retired, we identify all of the tasks that are running on that platform version revision in all Regions. We then send out one notification per account per Region, highlighting the affected tasks or services and a date when the retirements will start to take place.

The following illustration shows the lifecycle of a Fargate platform version revision from a new revision launch to the platform revision retirement.



The following information provides details.

- After a new platform version revision is launched, all new tasks are scheduled on this revision.
- Existing tasks that have been scheduled and running remain on the revision they were originally placed on for the duration of the task and aren't migrated to the new revision.
- New tasks, for example as part of an update to a service or Fargate task retirement, are placed on to latest platform version revision available at the time of launch.

Task retirement notifications are sent through AWS Health Dashboard as well as through an email to the registered email address and includes the following information:

- The task retirement date - The task is stopped on or after this date.
- For standalone tasks, the IDs of the tasks.
- For service tasks, the ID of the cluster where the service runs and the IDs of the service.
- The next steps you need to take.

Typically, we send one notification each for service and standalone tasks in each AWS Region. However, in certain cases you might receive more than one event for each task type, for example when there are too many tasks to be retired that will surpass limits in our notification mechanisms.

You can identify tasks scheduled for retirement in the following ways:

- The AWS Health Dashboard

  AWS Health notifications can be sent through Amazon EventBridge to archival storage such as Amazon Simple Storage Service, take automated actions such as run an AWS Lambda function, or other notification systems such as Amazon Simple Notification Service. For more information, see [Monitoring AWS Health events with Amazon EventBridge](). For sample configuration to send notifications to Amazon Chime, Slack, or Microsoft Teams, see the [AWS Health Aware]() repository on GitHub.

The following is a sample EventBridge event.

```
{
    "version": "0",
    "id": "3c268027-f43c-0171-7425-1d799EXAMPLE",
    "detail-type": "AWS Health Event",
    "source": "aws.health",
    "account": "123456789012",
    "time": "2023-08-16T23:18:51Z",
    "region": "us-east-1",
    "resources": [
        "cluster|service",
        "cluster|service"
    ],
    "detail": {
        "eventArn": "arn:aws:health:us-east-1::event/ECS/
AWS_ECS_TASK_PATCHING_RETIREMENT/AWS_ECS_TASK_PATCHING_RETIREMENT_test1",
        "service": "ECS",
        "eventScopeCode": "ACCOUNT_SPECIFIC",
        "communicationId":
 "7988399e2e6fb0b905ddc88e0e2de1fd17e4c9fa60349577446d95a18EXAMPLE",
        "lastUpdatedTime": "Wed, 16 Aug 2023 23:18:52 GMT",
        "eventRegion": "us-east-1",
        "eventTypeCode": "AWS_ECS_TASK_PATCHING_RETIREMENT",
        "eventTypeCategory": "scheduledChange",
        "startTime": "Wed, 16 Aug 2023 23:18:51 GMT",
        "endTime": "Fri, 18 Aug 2023 23:18:51 GMT",
        "eventDescription": [
            {
                "language": "en_US",
                "latestDescription": "\\nA software update has been deployed to
 Fargate which includes CVE patches or other critical patches. No action is required
 on your part. All new tasks launched automatically uses the latest software
 version. For existing tasks, your tasks need to be restarted in order for these
 updates to apply. Your tasks running as part of the following ECS Services will
 be automatically updated beginning Wed, 16 Aug 2023 23:18:51 GMT.\\n\\nAfter Wed,
 16 Aug 2023 23:18:51 GMT, the ECS scheduler will gradually replace these tasks,
 respecting the deployment settings for your service. Typically, services should
 see little to no interruption during the update and no action is required. When AWS
```

```
 stops tasks, AWS uses the minimum healthy percent (1) and launches a new task in
 an attempt to maintain the desired count for the service. By default, the minimum
 healthy percent of a service is 100 percent, so a new task is started first before
 a task is stopped. Service tasks are routinely replaced in the same way when
 you scale the service or deploy configuration changes or deploy task definition
 revisions. If you would like to control the timing of this restart you can update
 the service before Wed, 16 Aug 2023 23:18:51 GMT, by running the update-service
 command from the ECS command-line interface specifying force-new-deployment for
 services using Rolling update deployment type. For example:\\n\\n$ aws ecs update-
service -service service_name \\\n--cluster cluster_name -force-new-deployment\
\n\\nFor services using Blue/Green deployment type with AWS CodeDeploy:\\nPlease
 refer to create-deployment document (2) and create new deployment using same task
 definition revision.\\n\\nFor further details on ECS deployment types, please
 refer to ECS Deployment Developer Guide (1).\\nFor further details on Fargate's
 update process, please refer to the AWS Fargate User Guide (3).\\nIf you have
 any questions or concerns, please contact AWS Support (4).\\n\\n(1) https://
docs.aws.amazon.com/AmazonECS/latest/developerguide/deployment-types.html\\n(2)
 https://docs.aws.amazon.com/cli/latest/reference/deploy/create-deployment.html\\n(3)
 https://docs.aws.amazon.com/AmazonECS/latest/userguide/task-maintenance.html\\n(4)
 https://aws.amazon.com/support\\n\\nA list of your affected resources(s) can be
 found in the 'Affected resources' tab in the 'Cluster/ Service' format in the AWS
 Health Dashboard. \\n\\n"
            }
        ],
      "affectedEntities": [
                {
                    "entityValue": "arn:aws:ecs:eu-west-1:111222333444:task/
examplecluster/00805ce1d81940b5a37398e5a2c23333"
                },
                {
                    "entityValue": "arn:aws:ecs:eu-west-1:111222333444:task/
examplecluster/00805ce1d81940b5a37398e5a2c25555"
                }
            }
        ]
    }
}
```

- Email

  An email is sent to the registered email for the AWS account ID.

For information about how to prepare for task retirement, see Prepare for AWS Fargate task retirement on Amazon ECS .

Can I opt-out of task retirement?

No. As part of the AWS shared responsibility model, AWS is responsible for managing and maintaining the underlying infrastructure for AWS Fargate. This includes performing periodic platform updates to ensure security and stability. These updates are automatically applied by AWS and are not something customers can opt-out of. This is a key benefit of using a AWS Fargate compared to running your workloads on EC2 instances, the responsibility for maintaining the underlying platform is handled by AWS. This model allows you to focus on your applications rather than infrastructure maintenance. By automatically applying these platform updates, AWS is able to keep the Fargate environment up-to-date and secure, without any action required from you as the customer. This helps provide a reliable and secure containerized environment for running your workloads on Fargate.

Can I get task retirement notifications through other AWS services?

AWS sends a task retirement notification to the AWS Health Dashboard and to the primary email contact on the AWS account. The AWS Health Dashboard provides a number of integrations into other AWS services, including EventBridge. You can use EventBridge to automate the visibility of the notices (For example. forwarding the message to a ChatOps tool). For more information, see Solution overview: Capturing task retirement notifications.

Can I change a task retirement after it is scheduled?

No. The schedule is based off the task retirement wait time which has a default of 7 days. If you need more time, you can choose to configure the wait period to 14 days. For more information, see Step 2: Capture task retirement notifications to alert teams and take actions. The change in this configuration applies to retirements that will be scheduled in the future. Currently scheduled retirements are not impacted. If you have any further concerns, contact Support.

How does Amazon ECS handle tasks that are part of a service?

For service tasks, you do not need to take any action in response to task retirement unless you want to replace these tasks before AWS does. When the Amazon ECS scheduler stops the tasks, it uses the minimum healthy percent and launches a new task in an attempt to maintain the desired count for the service. To minimalize the impact of Fargate task retirement, workloads should be deployed following Amazon ECS best practices. For example, when deploying a stateless application as an Amazon ECS service, such as a web or API server, customers should deploy multiple task replicas and set the minimumHealthyPercent to 100%. By default, the minimum healthy percent of a service is 100 percent. Therefore, when Fargate starts retiring tasks, Amazon ECS first schedules a new task and waits for it to be running, before retiring an old task. Service tasks are routinely replaced as part of task retirement in the same way when you scale the service, deploy configuration changes, or deploy task definition revisions. To prepare for the task retirement process, see Prepare for AWS Fargate task retirement on Amazon ECS .

Can Amazon ECS automatically handle standalone tasks?

No. AWS can't create a replacement task for standalone tasks which are started by `RunTask`, scheduled tasks (for example through EventBridge Scheduler), AWS Batch, or AWS Step Functions. Amazon ECS manages only tasks that are part of a service.

Troubleshooting service availability during task retirement

If Amazon ECS cannot start a replacement task during task retirement, your service availability may be impacted. This can occur due to incorrect customer configuration, such as:

- Missing or incorrectly configured IAM roles
- Insufficient capacity in the target subnets
- Security group misconfigurations
- Task definition errors

When Amazon ECS cannot launch replacement tasks, the retired tasks are stopped without replacement, reducing your service's available capacity and potentially causing service disruption. Monitor your service's task count and Amazon CloudWatch metrics to ensure replacement tasks are successfully launched during retirement events.

Prepare for AWS Fargate task retirement on Amazon ECS

In order to prepare for task retirement, perform the following operations:

1. Set the task retirement wait period.
2. Capture task retirement notifications to notify team members.
3. You can't control the exact timing of a task retirement, however, you can control the replacement of tasks by updating the service with the force-deployment option.

Step 1: Set the task wait time

You can configure the time that Fargate starts the task retirement. For workloads that require immediate application of the updates, choose the immediate setting (`0`). When you need more control, for example, when a task can only be stopped during a certain window, configure the 7 day (7), or 14 day (`14`) option.

We recommend that you choose a shorter waiting period in order to pick up newer platform versions revisions sooner.

Configure the wait period by running `put-account-setting-default` or `put-account-setting` as the root user or an administrative user. Use the `fargateTaskRetirementWaitPeriod` option for the `name` and the `value` option set to one of the following values:

- `0` - AWS sends the notification, and immediately starts to retire the affected tasks.
- `7` - AWS sends the notification, and waits 7 calendar days before starting to retire the affected tasks.
- `14` - AWS sends the notification, and waits 14 calendar days before starting to retire the affected tasks.

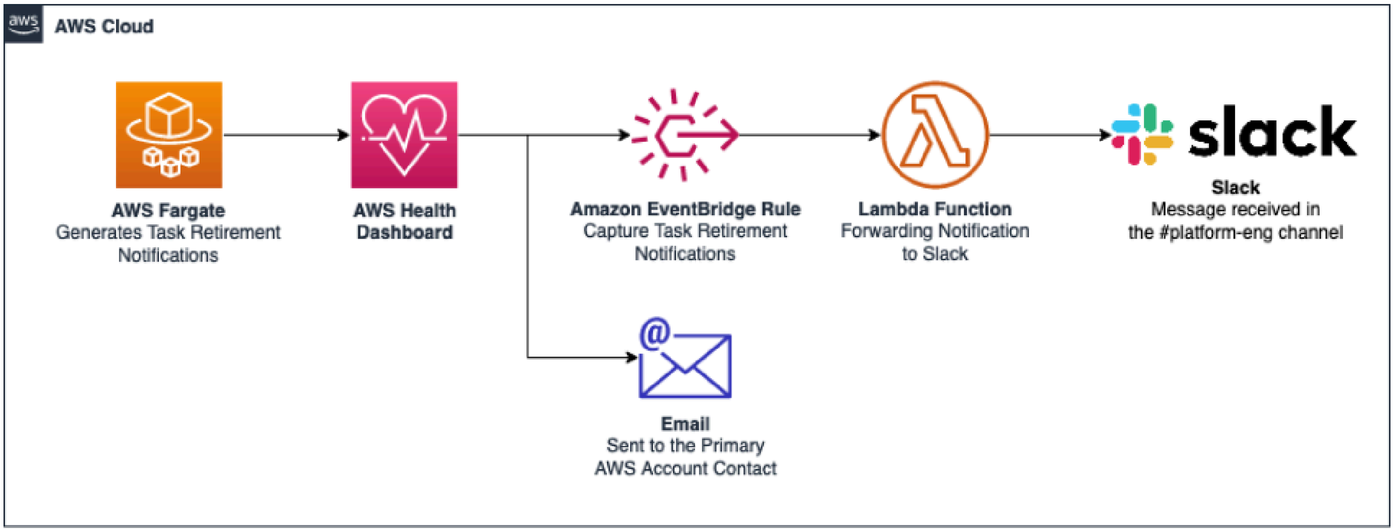
The default is 7 days.

For more information, see, put-account-setting-default and put-account-setting in the *Amazon Elastic Container Service API Reference*.

Step 2: Capture task retirement notifications to alert teams and take actions

When there is an upcoming task retirement, AWS sends a task retirement notification to the AWS Health Dashboard, and to the primary email contact on the AWS account. The AWS Health Dashboard provides a number of integrations into other AWS services, including Amazon EventBridge. You can use EventBridge to build automations from a task retirement notification, such as increasing the visibility of the upcoming retirement by forwarding the message to a ChatOps tool. AWS Health Aware is a resource that shows the power of the AWS Health Dashboard and how notifications can be distributed throughout an organization. You can forward a task retirement notification to a chat application, such as Slack.

The following illustration shows the solution overview.



The following information provides details.

- Fargate sends the task retirement notification to the AWS Health Dashboard.
- The AWS Health Dashboard sends mail to the primary email contact on the AWS account, and notifies EventBridge.
- EventBridge has a rule that captures the retirement notification.

  The rule looking for events with the Event Detail Type: `"AWS Health Event" and the Event Detail Type Code: "AWS_ECS_TASK_PATCHING_RETIREMENT"`
- The rule triggers a Lambda function that forwards the information to Slack using a Slack Incoming Webhook. For more information, see [Incoming Webhooks](#).

For a code example, see [Capturing AWS Fargate Task Retirement Notifications](#) on Github.

Step 3: Control the replacement of tasks

You can't control the exact timing of a task retirement, however, you can define a wait time. If you want control over replacing tasks at your own schedule, you can capture the task retirement notice to first understand the task retirement date. You can then redeploy your service to launch replacement tasks, and likewise replace any standalone tasks.For services that use rolling deployment, you update the service using `update-service` with the `force-deployment` option before the retirement start time.

The following `update-service` example uses the `force-deployment` option.

```
aws ecs update-service --service service_name \
    --cluster cluster_name \
     --force-new-deployment
```

For services that use the blue/green deployment, you need to create a new deployment in AWS CodeDeploy. For information about how to create the deployment, see create-deployment in the *AWS Command Line Interface Reference*.

Supported Regions for Amazon ECS on AWS Fargate

You can use the following tables to verify the Region support for Linux containers on AWS Fargate and Windows containers on AWS Fargate.

Linux containers on AWS Fargate

Amazon ECS Linux containers on AWS Fargate are supported in the following AWS Regions. The supported Availability Zone IDs are noted when applicable.

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1 (`usw1-az1` & `usw1-az3` only)
US West (Oregon)	us-west-2
Canada (Central)	ca-central-1
Canada West (Calgary)	ca-west-1
Mexico (Central)	mx-central-1
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1

Region Name	Region
Asia Pacific (Tokyo)	ap-northeast-1 (`apne1-az1` , `apne1-az2` , & `apne1-az4` only)
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Thailand)	ap-southeast-7
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Malaysia)	ap-southeast-5
Canada (Central)	ca-central-1
Canada West (Calgary)	ca-west-1
China (Beijing)	cn-north-1 (`cnn1-az1` & `cnn1-az2` only)
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1

Region Name	Region
Europe (Spain)	eu-south-2
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
Israel (Tel Aviv)	il-central-1
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

Windows containers on AWS Fargate

Amazon ECS Windows containers on AWS Fargate are supported in the following AWS Regions. The supported Availability Zone IDs are noted when applicable.

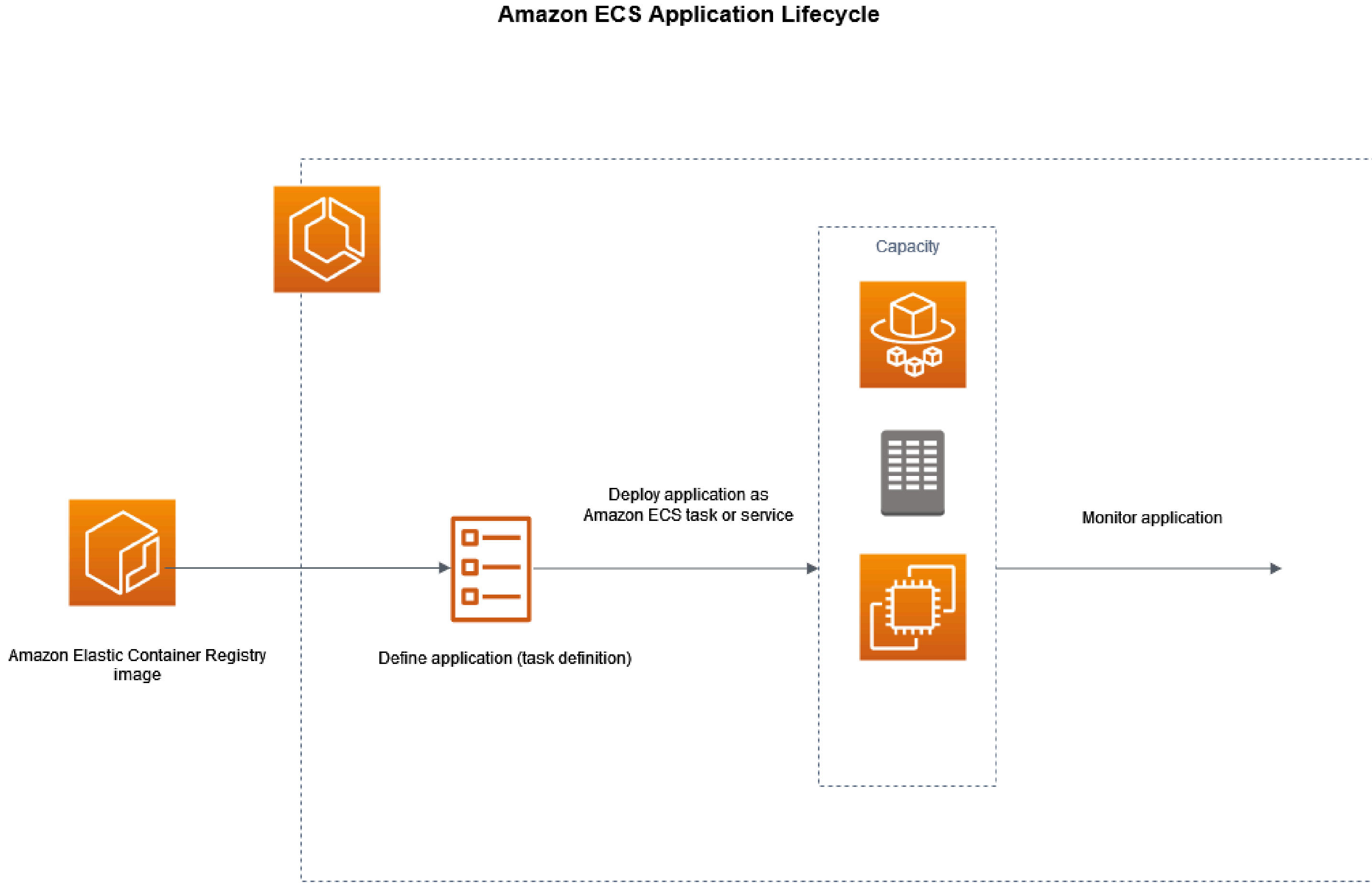
Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1 (`use1-az1`, `use1-az2`, `use1-az4`, `use1-az5`, & `use1-az6`only)
US West (N. California)	us-west-1 (`usw1-az1` & `usw1-az3` only)
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1

Region Name	Region
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Malaysia)	ap-southeast-5
Asia Pacific (Tokyo)	ap-northeast-1 (`apne1-az1` , `apne1-az2` , & `apne1-az4` only)
Canada (Central)	ca-central-1 (`cac1-az1` & `cac1-az2` only)
Canada West (Calgary)	ca-west-1
China (Beijing)	cn-north-1 (`cnn1-az1` & `cnn1-az2` only)
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Stockholm)	eu-north-1

Region Name	Region
South America (São Paulo)	sa-east-1
Israel (Tel Aviv)	il-central-1
Middle East (UAE)	me-central-1
Middle East (Bahrain)	me-south-1

Architect your solution for Amazon ECS

The following diagram shows the application lifecycle and how it works with the Amazon ECS components.



You must architect your applications so that they can run on *containers*. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an *image*. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that contains the instructions for building a container. After they're built, these images are stored in a *registry* such as Amazon ECR where they can be downloaded from.

After you create and store your image, you create an Amazon ECS task definition. A *task definition* is a blueprint for your application. It is a text file in JSON format that describes the parameters and one or more containers that form your application. For example, you can use it to specify the image and parameters for the operating system, which containers to use, which ports to open for your application, and what data volumes to use with the containers in the task. The specific parameters available for your task definition depend on the needs of your specific application.

After you define your task definition, you deploy it as either a service or a task on your cluster. A *cluster* is a logical grouping of tasks or services that runs on the capacity infrastructure that is registered to a cluster.

A *task* is the instantiation of a task definition within a cluster. You can run a standalone task, or you can run a task as part of a service. You can use an Amazon ECS *service* to run and maintain your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

The *container agent* runs on each container instance within an Amazon ECS cluster. The agent sends information about the current running tasks and resource utilization of your containers to Amazon ECS. It starts and stops tasks whenever it receives a request from Amazon ECS.

After you deploy the task or service, you can use any of the following tools to monitor your deployment and application:

- CloudWatch
- Runtime Monitoring

Capacity

The capacity is the infrastructure where your containers run. The following are the options:

- Amazon EC2 instances
- Serverless (AWS Fargate)
- On-premises virtual machines (VM) or servers

You specify the infrastructure when you create a cluster. You also specify the infrastructure type when you register a task definition. The task definition refers to the infrastructure as the "launch type". You also use the launch type when you run a standalone task or deploy a service. For information about the launch type options, see [Amazon ECS launch types](#).

Service endpoints

The service endpoint is the URL of the entry point for Amazon ECS that you use to connect to the service programmatically using either Internet Protocol version 4 (IPv4) or Internet Protocol version 6 (IPv6). By default, requests that you make to connect to Amazon ECS programmatically use service endpoints that support only IPv4 requests. To connect programmatically with Amazon ECS using either IPv4 or IPv6 requests, you can use a dual-stack endpoint. For information about using a dual-stack endpoint, see Using Amazon ECS dual-stack endpoints.

Networking

AWS resources are created in subnets. When you use EC2 instances, Amazon ECS launches the instances in the subnet that you specify when you create a cluster. Your tasks run in the instance subnet. For Fargate or on-premises virtual machines, you specify the subnet when you run a task or create a service.

Depending on your application, the subnet can be a private or public subnet and the subnet can be in any of the following AWS resources:

- Availability Zones
- Local Zones
- Wavelength Zones
- AWS Regions
- AWS Outposts

For more information, see Amazon ECS applications in shared subnets, Local Zones, and Wavelength Zones or Amazon Elastic Container Service on AWS Outposts.

You can have your application connect to the internet by using one of the following methods:

- A public subnet with an internet gateway

  Use public subnets when you have public applications that require large amounts of bandwidth or minimal latency. Applicable scenarios include video streaming and gaming services.

- A private subnet with a NAT gateway

Use private subnets when you want to protect your containers from direct external access. Applicable scenarios include payment processing systems or containers storing user data and passwords.

- AWS PrivateLink

  Use AWS PrivateLink to have private connectivity between VPCs, AWS services, and your on-premises networks without exposing your traffic to the public internet.

Feature access

You can use your Amazon ECS account settings to access the following features:

- Container Insights

  CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. The metrics include utilization for resources such as CPU, memory, disk, and network.
- `awsvpc` trunking

  For certain EC2 instances types, you can have additional network interfaces (ENIs) available on newly launched container instances.
- Tagging authorization

  Users must have permissions for actions that create a resource, such as `ecsCreateCluster`. If tags are specified in the resource-creating action, AWS performs additional authorization on the `ecs:TagResource` action to verify if users or roles have permissions to create tags.
- Fargate FIPS-140 compliance

  Fargate supports the Federal Information Processing Standard (FIPS-140) which specifies the security requirements for cryptographic modules that protect sensitive information. It is the current United States and Canadian government standard, and is applicable to systems that are required to be compliant with Federal Information Security Management Act (FISMA) or Federal Risk and Authorization Management Program (FedRAMP).
- Fargate task retirement time changes

  You can configure the wait period before Fargate tasks are retired for patching.
- Dual stack VPC

Allow tasks to communicate over IPv4, IPv6, or both.

- Amazon Resource Name (ARN) format

  Certain features, such as tagging authorization, require a new Amazon Resource Name (ARN) format.

For more information, see Access Amazon ECS features with account settings.

IAM roles

An IAM role is an IAM identity that you can create in your account that has specific permissions. In Amazon ECS, you can create roles to grant permissions to Amazon ECS resource such as containers or services.

Some Amazon ECS features require roles. For more information, see IAM roles for Amazon ECS.

Logging

Logging and monitoring are important aspects of maintaining the reliability, availability, and performance of Amazon ECS workloads. The following options are available:

- Amazon CloudWatch logs - route logs to Amazon CloudWatch
- FireLens for Amazon ECS - route logs to an AWS service or AWS Partner Network destination for log storage and analysis. The AWS Partner Network is a global community of partners that leverages programs, expertise, and resources to build, market, and sell customer offerings.

Amazon ECS launch types

The task definition launch type defines what capacity the task can run on, for example AWS Fargate.

After you choose the launch type, Amazon ECS verifies that the task definition parameters you configure work with the launch type.

The following table describes the available launch types.

Launch type		Learn more
Fargate	Fargate is a serverless, pay-as-you-go compute engine that lets you focus on building applications without managing servers.	Fargate launch type for Amazon ECS
EC2	You choose the instance type, the number of instances, and manage the capacity.	EC2 launch type for Amazon ECS
On-premises virtual machines (VM) or servers	Amazon ECS Anywhere provides support for registeri ng an external instance such as an on-premises server or virtual machine (VM), to your cluster.	External (Amazon ECS Anywhere) launch type for Amazon ECS

Fargate launch type for Amazon ECS

Fargate is a serverless, pay-as-you-go compute engine that lets you focus on building applications without managing servers. When you choose Fargate, you don't need to manage an EC2 infrastructure. All you need to do is build your container image and define which cluster you want to run your applications on. Fargate has native integration with AWS services including:

- Amazon VPC
- Auto Scaling
- Elastic Load Balancing
- IAM
- Secrets Manager

You have more control with Fargate than EC2 because you select the exact CPU and memory that your application needs. Fargate handles scaling out your capacity, so you don't need to worry about spikes in traffic. This means that there is less operational effort with Fargate.

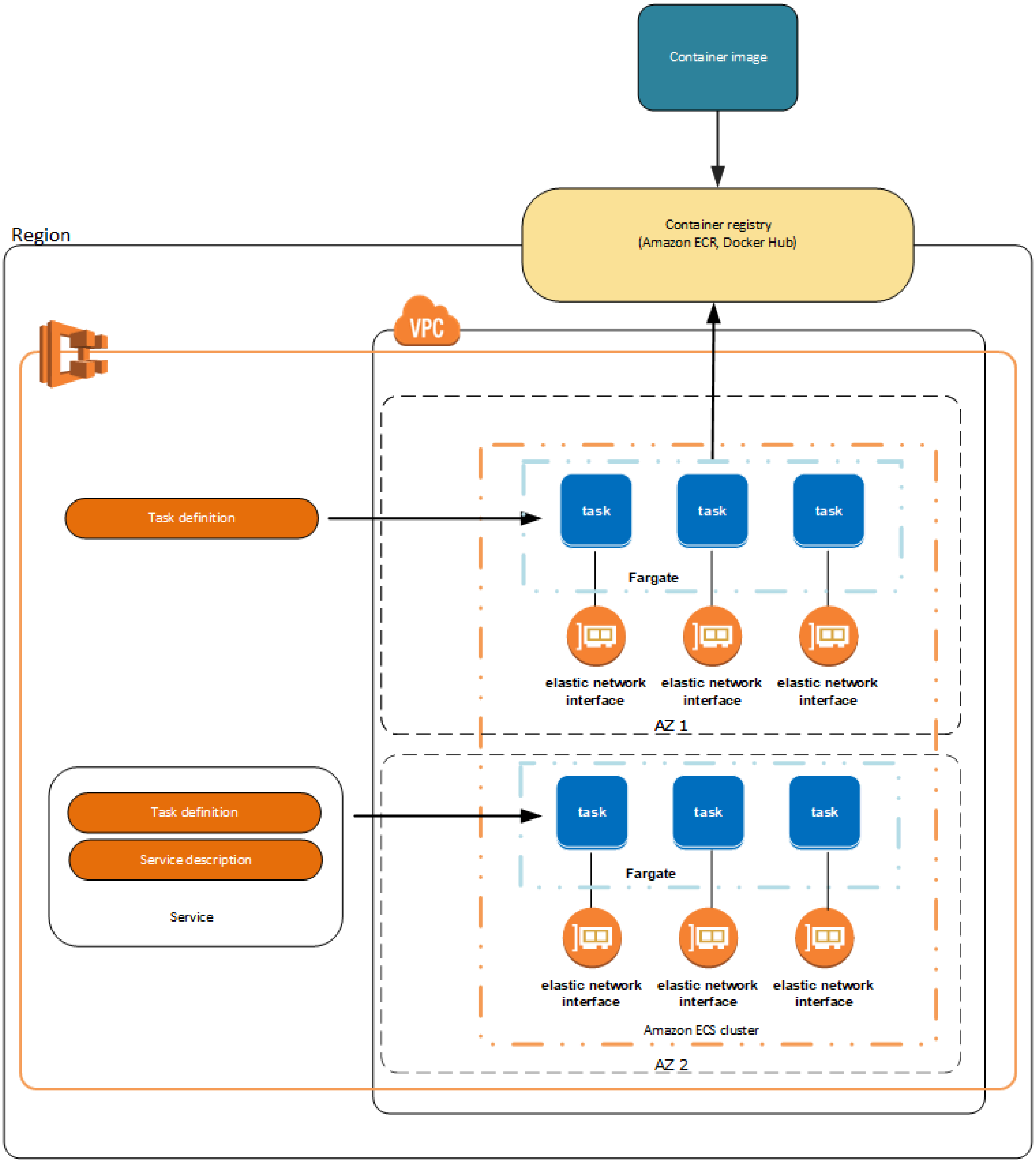
Fargate meets the standards for compliance programs including PCI, FIPS 140-3, FedRAMP, and HIPAA. For more information see AWS Services in Scope by Compliance Program .

Fargate is suitable for the following workloads:

- Large workloads that require low operational overhead
- Small workloads that have occasional burst
- Tiny workloads
- Batch workloads

For information about the Regions that support Fargate, see the section called “AWS Fargate Regions”.

The following diagram shows the general architecture.



For more information about Amazon ECS on Fargate, see [AWS Fargate for Amazon ECS](#).

Linux containers on Fargate container image pull behavior for Amazon ECS

Every Fargate task runs on its own single use, single tenant instance. When you run Linux containers on Fargate, container images or container image layers are not cached on the instance. Therefore, for each container image defined in the task, the whole container image needs to be pulled from the container image registry for each Fargate task. The time it takes to pull the images is directly correlated to the time taken to start an Fargate task.

Take the following into account to optimize the image pull time.

Container image proximity

To reduce the time it takes to download container images, locate the data as close to the compute as possible. Pulling a container image over the internet or across AWS Regions might impact the download time. We recommend that you store the container image in the same Region where the task will run. If you store the container image in Amazon ECR, use a VPC interface endpoint to further reduce the image pull time. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) in the *Amazon ECR User Guide*.

Container image size reduction

The size of a container image directly impacts the download time. Reducing the size of the container image or the number of container image layers, can reduce the time it takes for an image to download. Lightweight base images (such as the minimal Amazon Linux 2023 container image) can be significantly smaller than those based on traditional operating system base images. For more information about the minimal image, see AL2023 Minimal container image in the *Amazon Linux 2023 User Guide*.

Alternative compression algorithms

Container image layers are often compressed when pushed to a container image registry. Compressing the container image layer reduces the amount of data that has to be transferred across the network and stored in the container image registry. After a container image layer has been downloaded to an instance by the container runtime, that layer is decompressed. The compression algorithm used and the amount of vCPUs available to the runtime impact the time it takes to decompress the container image. On Fargate, you can increase the size of the task or leverage the more performant zstd compression algorithm to reduce the time taken for decompression. For more information, see zstd on GitHub. For information about how to implement the images for Fargate, see Reducing AWS Fargate Startup Times with zstd Compressed Container Images.

Lazy Loading container images

For large container images (> 250mb), it might be optimal to lazy load a container image rather than downloading all of the container image. On Fargate, you can use Seekable OCI (SOCI) to lazy load a container image from a container image registry. For more information, see soci-snapshotter on GitHub and Lazy loading container images using Seekable OCI (SOCI).

Windows containers on Fargate container image pull behavior for Amazon ECS

Fargate Windows caches the most recent month's, and the previous month's, servercore base image provided by Microsoft. These images match the KB/Build number patches updated each Patch Tuesday. For example, on April 9, 2024, Microsoft released KB5036896 (17763.5696) for Windows Server 2019. The previous month KB on March 12, 2024 was KB5035849 (17763.5576). So for the platforms `WINDOWS_SERVER_2019_CORE` and `WINDOWS_SERVER_2019_FULL` the following container images were cached::

- `mcr.microsoft.com/windows/servercore:ltsc2019`
- `mcr.microsoft.com/windows/servercore:10.0.17763.5696`
- `mcr.microsoft.com/windows/servercore:10.0.17763.5576`

Additionally, on April 9, 2024, Microsoft released KB5036909 (20348.2402) for Windows Server 2022. The previous month KB on March 12, 2024 was KB5035857 (20348.2340). So for the platforms `WINDOWS_SERVER_2022_CORE` and `WINDOWS_SERVER_2022_FULL` the following container images were cached:

- `mcr.microsoft.com/windows/servercore:ltsc2022`
- `mcr.microsoft.com/windows/servercore:10.0.20348.2402`
- `mcr.microsoft.com/windows/servercore:10.0.20348.2340`

EC2 launch type for Amazon ECS

The EC2 launch type is suitable for large workloads that must be price optimized.

When considering how to model task definitions and services using the EC2 launch type, we recommend that you consider what processes must run together and how you might go about scaling each component.

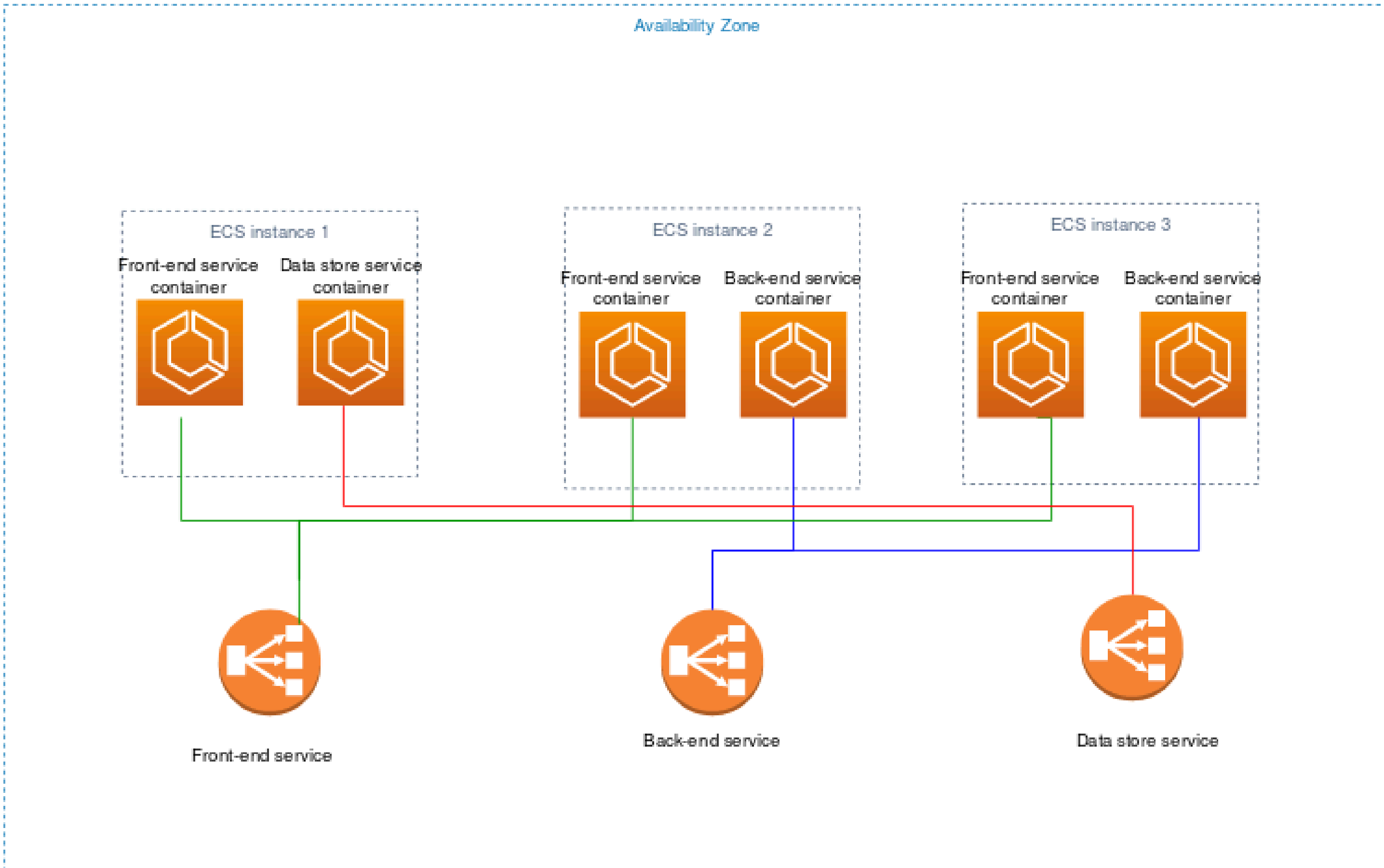
As an example, suppose that an application consists of the following components:

- A frontend service that displays information on a webpage
- A backend service that provides APIs for the frontend service
- A data store

For this example, create task definitions that group the containers that are used for a common purpose together. Separate the different components into multiple and separate task definitions. The following example cluster has three container instances that are running three front-end service containers, two backend service containers, and one data store service container.

You can group related containers in a task definition, such as linked containers that must be run together. For example, add a log streaming container to your front-end service and include it in the same task definition.

After you have your task definitions, you can create services from them to maintain the availability of your desired tasks. For more information, see Creating an Amazon ECS rolling update deployment. In your services, you can associate containers with Elastic Load Balancing load balancers. For more information, see Use load balancing to distribute Amazon ECS service traffic. When your application requirements change, you can update your services to scale the number of desired tasks up or down. Or, you can update your services to deploy newer versions of the containers in your tasks. For more information, see Updating an Amazon ECS service.



Container image pull behavior for the EC2 and external launch types for Amazon ECS

The time that it takes a container to start up varies, based on the underlying container image. For example, a fatter image (full versions of Debian, Ubuntu, and Amazon1/2) might take longer to start up because there are a more services that run in the containers compared to their respective slim versions (Debian-slim, Ubuntu-slim, and Amazon-slim) or smaller base images (Alpine).

When the Amazon ECS agent starts a task, it pulls the Docker image from its remote registry, and then caches a local copy. When you use a new image tag for each release of your application, this behavior is unnecessary.

The `ECS_IMAGE_PULL_BEHAVIOR` agent parameter determines the image pull behavior. The following options are available:

- `ECS_IMAGE_PULL_BEHAVIOR`: `default`

  The image will be pulled remotely. If the pull fails, the cached image in the instance is used.

- `ECS_IMAGE_PULL_BEHAVIOR`: `always`

  The image will be pulled remotely. If the pull fails, the task fails.

To speed up deployment, set the Amazon ECS agent parameter to one of the following values:

- `ECS_IMAGE_PULL_BEHAVIOR`: `once`

  The image is pulled remotely only if it wasn't pulled by a previous task on the same container instance or if the cached image was removed by the automated image cleanup process. Otherwise, the cached image on the instance is used. This ensures that no unnecessary image pulls are attempted.
- `ECS_IMAGE_PULL_BEHAVIOR`: `prefer-cached`

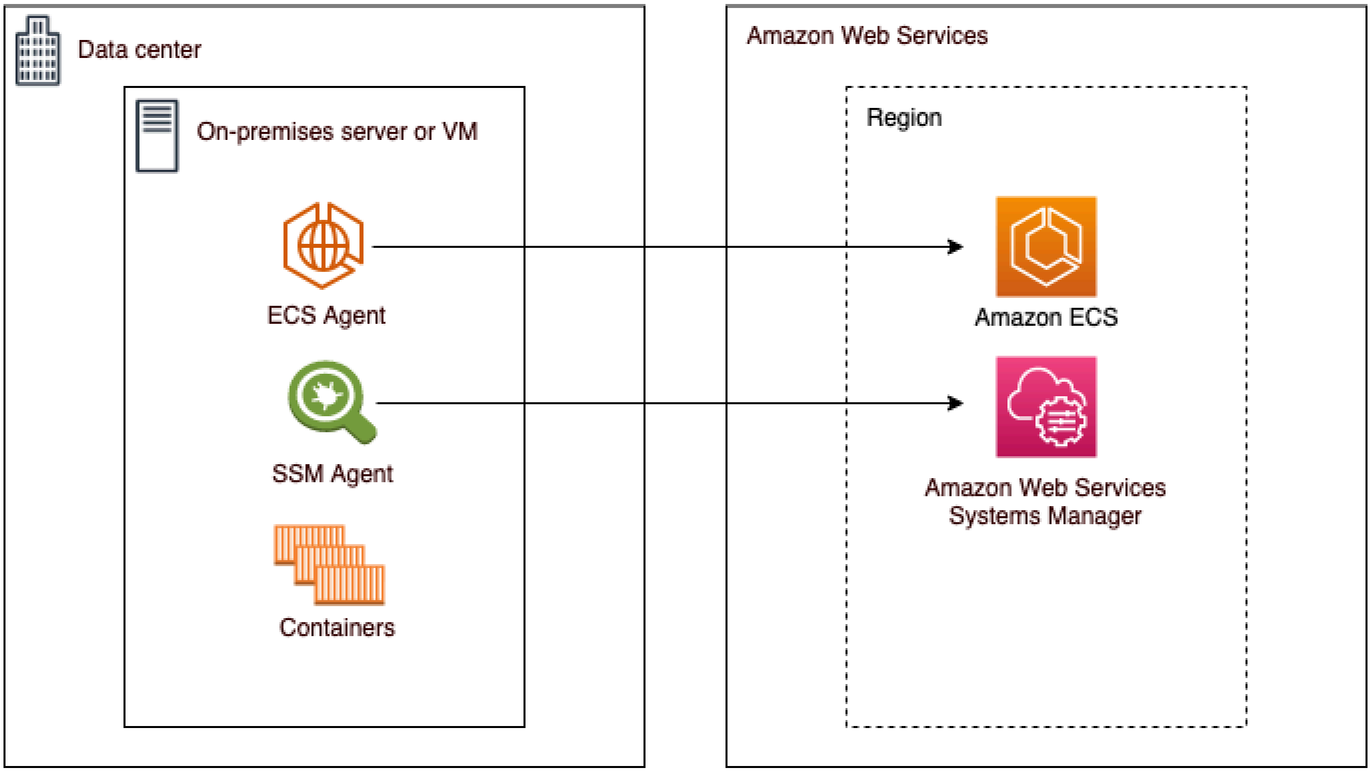
  The image is pulled remotely if there is no cached image. Otherwise, the cached image on the instance is used. Automated image cleanup is turned off for the container to ensure that the cached image isn't removed.

Setting the `ECS_IMAGE_PULL_BEHAVIOR` parameter to either of the preceding values can save time because the Amazon ECS agent uses the existing downloaded image. For larger Docker images, the download time might take 10-20 seconds to pull over the network.

External (Amazon ECS Anywhere) launch type for Amazon ECS

Amazon ECS Anywhere provides support for registering an *external instance* such as an on-premises server or virtual machine (VM), to your Amazon ECS cluster. External instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, the lack of Elastic Load Balancing support makes running these workloads less efficient. Amazon ECS added a new `EXTERNAL` launch type that you can use to create services or run tasks on your external instances.

The following provides a high-level system architecture overview of Amazon ECS Anywhere. Your on-premises server has both the Amazon ECS agent and the SSM agent installed.



For more information, see [Amazon ECS clusters for the external launch type](#).

Using Amazon ECS dual-stack endpoints

Amazon ECS dual-stack endpoints support requests to Amazon ECS over Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). For a list of all Amazon ECS endpoints, see [Amazon ECS endpoints and quotas](#) in the AWS General Reference.

When using the REST API, you directly access an Amazon ECS endpoint by using the endpoint name (URI). Amazon ECS supports only regional dual-stack endpoint names, which means that you must specify the region as part of the name.

Use the following naming convention for the dual-stack endpoint names: `ecs.region.api.aws`.

When using the AWS Command Line Interface (AWS CLI) and AWS SDKs, you can use a parameter or a flag to change to a dual-stack endpoint. You can also specify the dual-stack endpoint directly as an override of the Amazon ECS endpoint in the config file.

The following sections describe how to use dual-stack endpoints from the AWS CLI, the AWS SDKs, and the REST API.

Topics

- Using dual-stack endpoints from the AWS CLI
- Using dual-stack endpoints from the AWS SDKs
- Using dual-stack endpoints from the REST API

Using dual-stack endpoints from the AWS CLI

This section provides examples of AWS CLI commands used to make requests to a dual-stack endpoint. For more information about installing the AWS CLI or updating to the latest version, see Installing or updating to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

To use a dual-stack endpoint, you can set the configuration value `use_dualstack_endpoint` to `true` in the `config` file for the AWS CLI to direct all Amazon ECS requests made by the `ecs` AWS CLI command to the dual-stack endpoint for the specified region. You can specify the region in the `config` file or in a command by using the `--region` option. For more information about configuration files for the AWS CLI, see Configuration and credential file settings in the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

If you want to use a dual-stack endpoint for specific AWS CLI commands, you can use either of the following methods:

- You can use the dual-stack endpoint per command by setting the `--endpoint-url` parameter to `https://ecs.`*`aws-region`*`.api.aws` or `http://ecs.`*`aws-region`*`.api.aws` for any `ecs` command.

  The following example command lists all available clusters and uses the dual-stack endpoint for the request.

  ```
  $ aws ecs list-clusters --endpoint-url https://ecs.aws-region.api.aws
  ```
- You can set up separate profiles in your AWS Config file. For example, create one profile that sets `use_dualstack_endpoint` to `true` and a profile that does not set `use_dualstack_endpoint`. When you run a command, specify which profile you want to use, depending upon whether or not you want to use the dual-stack endpoint.

Using dual-stack endpoints from the AWS SDKs

This section provides examples of how to access a dual-stack endpoint by using the AWS SDKs.

AWS SDK for Java 2.x

The following example shows how to specify a dual-stack endpoint for the `us-east-1` Region using the AWS SDK for Java 2.x.

```
Region region = Region.US_EAST_1
EcsClient client =
 EcsClient.builder().region(region).dualstackEnabled(true).build();
```

AWS SDK for Go

The following example shows how to specify a dual-stack endpoint for the `us-east-1` Region using the AWS SDK for Go.

```
sess := session.Must(session.NewSession())
svc := ecs.New(sess, &aws.Config{
    Region: aws.String(endpoints.UsEast1RegionID),
    Endpoint: aws.String("https://ecs.us-east-1.api.aws")
})
```

For more information, see [Dual-stack and FIPS endpoints] in the *AWS SDKs and Tools Reference Guide*.

Using dual-stack endpoints from the REST API

When using the REST API, you can directly access a dual-stack endpoint by specifying it in your request. The following example uses the dual-stack endpoint to list all Amazon ECS clusters in the `us-east-1` Region.

```
POST / HTTP/1.1
Host: ecs.us-east-1.api.aws
Accept-Encoding: identity
Content-Length: 2
X-Amz-Target: AmazonEC2ContainerServiceV20141113.ListClusters
X-Amz-Date: 20150429T170621Z
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{}
```

Amazon ECS applications in shared subnets, Local Zones, and Wavelength Zones

Amazon ECS supports workloads that use Local Zones, Wavelength Zones, and AWS Outposts for when low latency or local data processing is a requirement.

- You can use Local Zones as an extension of an AWS Region to place resources in multiple locations closer to your end users.
- You can use Wavelength Zones to build applications that deliver ultra-low latencies to 5G devices and end users. Wavelength deploys standard AWS compute and storage services to the edge of telecommunication carriers' 5G networks.
- AWS Outposts brings native AWS services, infrastructure, and operating models to virtually any data center, co-location space, or on-premises facility.

> **Important**
>
> Amazon ECS on AWS Fargate workloads aren't supported in Local Zones, Wavelength Zones, or on AWS Outposts at this time.

For information about the differences between Local Zones, Wavelength Zones, and AWS Outposts , see How should I think about when to use AWS Wavelength, AWS Local Zones, or AWS Outposts for applications requiring low latency or local data processing in the AWS Wavelength FAQs.

Shared subnets

You can use *VPC sharing* to share subnets with other AWS accounts within the same AWS Organizations.

You can use shared VPCs for the EC2 launch type with the following considerations:

- The owner of the VPC subnet must share a subnet with a participant account before that account can use it for Amazon ECS resources.
- You can't use the VPC default security group for your container instances because it belongs to the owner. Additionally, participants can't launch instances using security groups that are owned by other participants or the owner.

- In a shared subnet, the participant and the owner separately controls the security groups within each respective account. The subnet owner can see security groups that are created by the participants but cannot perform any actions on them. If the subnet owner wants to remove or modify these security groups, the participant that created the security group must take the action.
- The shared VPC owner cannot view, update or delete a cluster that a participant creates in the shared subnet. This is in addition to the VPC resources that each account has different access to. For more information, see Responsibilities and permissions for owners and participants in the *Amazon VPC User Guide*.

You can use shared VPCs for the Fargate launch type with the following considerations::

- The owner of the VPC subnet must share a subnet with a participant account before that account can use it for Amazon ECS resources.
- You can't create a service or run a task using the default security group for the VPC because it belongs to the owner. Additionally, participants can't create a service or run a task using security groups that are owned by other participants or the owner.
- In a shared subnet, the participant and the owner separately controls the security groups within each respective account. The subnet owner can see security groups that are created by the participants but cannot perform any actions on them. If the subnet owner wants to remove or modify these security groups, the participant that created the security group must take the action.
- The shared VPC owner cannot view, update or delete a cluster that a participant creates in the shared subnet. This is in addition to the VPC resources that each account has different access to. For more information, see Responsibilities and permissions for owners and participants in the *Amazon VPC User Guide*.

For more information about VPC subnet sharing, see Share your VPC with other accounts in the *Amazon VPC User Guide*.

Local Zones

A *Local Zone* is an extension of an AWS Region in close geographic proximity to your users. Local Zones have their own connections to the internet and support AWS Direct Connect. Resources that are created in a Local Zone can serve local users with low-latency communications. For more information, see AWS Local Zones.

A Local Zone is represented by a Region code followed by an identifier that indicates the location (for example, `us-west-2-lax-1a`).

To use a Local Zone, you must opt in to the zone. After you opt in, you must create an Amazon VPC and subnet in the Local Zone.

You can launch Amazon EC2 instances, Amazon FSx file servers, and Application Load Balancers to use for your Amazon ECS clusters and tasks.

For more information, see What is AWS Local Zones? in the *AWS Local Zones User Guide*.

Wavelength Zones

You can use *AWS Wavelength* to build applications that deliver ultra-low latency to mobile devices and end users. Wavelength deploys standard AWS compute and storage services to the edge of telecommunication carriers' 5G networks. You can extend an Amazon Virtual Private Cloud to one or more Wavelength Zones. Then, you can use AWS resources such as Amazon EC2 instances to run applications that require ultra-low latency and a connection to AWS services in the Region.

A Wavelength Zone is an isolated Zone in the carrier location where the Wavelength infrastructure is deployed. Wavelength Zones are tied to an AWS Region. A Wavelength Zone is a logical extension of a Region, and is managed by the control plane in the Region.

A Wavelength Zone is represented by a Region code followed by an identifier that indicates the Wavelength Zone (for example, `us-east-1-wl1-bos-wlz-1`).

To use a Wavelength Zone, you must opt in to the Zone. After you opt in, you must create an Amazon VPC and subnet in the Wavelength Zone. Then, you can launch your Amazon EC2 instances in the Zone to use for your Amazon ECS clusters and tasks.

For more information, see Get started with AWS Wavelength in the *AWS Wavelength Developer Guide*.

Wavelength Zones aren't available in all AWS Regions. For information about the Regions that support Wavelength Zones, see Available Wavelength Zones in the *AWS Wavelength Developer Guide*.

Amazon Elastic Container Service on AWS Outposts

AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud.

Amazon ECS on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications.

For more information about AWS Outposts, see the *AWS Outposts User Guide*.

Considerations

The following are considerations of using Amazon ECS on AWS Outposts:

- Amazon Elastic Container Registry, AWS Identity and Access Management, and Network Load Balancer run in the AWS Region, not on AWS Outposts. This will increase latencies between these services and the containers.
- AWS Fargate is not available on AWS Outposts.

The following are network connectivity considerations for AWS Outposts:

- If network connectivity between your AWS Outposts and its AWS Region is lost, your clusters will continue to run. However, you cannot create new clusters or take new actions on existing clusters until connectivity is restored. In case of instance failures, the instance will not be automatically replaced. The CloudWatch Logs agent will be unable to update logs and event data.
- We recommend that you provide reliable, highly available, and low latency connectivity between your AWS Outposts and its AWS Region.

Prerequisites

The following are prerequisites for using Amazon ECS on AWS Outposts:

- You must have installed and configured an Outpost in your on-premises data center.
- You must have a reliable network connection between your Outpost and its AWS Region.

Overview of cluster creation on AWS Outposts

The following is an overview of the configuration:

1. Create a role and policy with rights on AWS Outposts.
2. Create an IAM instance profile with rights on AWS Outposts.
3. Create a VPC, or use an existing one that is in the same Region as your AWS Outposts.
4. Create a subnet or use an existing one that is associated with the AWS Outposts.

   This is the subnet where the container instances run.
5. Create a security group for the container instances in your cluster.
6. Create an Amazon ECS cluster.
7. Define the Amazon ECS container agent environment variables to launch the instance into the cluster.
8. Run a container.

For detailed information about how to integrate Amazon ECS with AWS Outposts, see Extend Amazon ECS across two AWS Outposts racks.

The following example creates an Amazon ECS cluster on an AWS Outposts.

1. Create a role and policy with rights on AWS Outposts.

   The `role-policy.json` file is the policy document that contains the effect and actions for resources. For information about the file format, see PutRolePolicy in the *IAM API Reference*

   ```
   aws iam create-role --role-name ecsRole \
       --assume-role-policy-document file://ecs-policy.json
   aws iam put-role-policy --role-name ecsRole --policy-name ecsRolePolicy \
       --policy-document file://role-policy.json
   ```

2. Create an IAM instance profile with rights on AWS Outposts.

   ```
   aws iam create-instance-profile --instance-profile-name outpost
   aws iam add-role-to-instance-profile --instance-profile-name outpost \
       --role-name ecsRole
   ```

3. Create a VPC.

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16
```

4. Create a subnet associated with your AWS Outposts.

```
aws ec2 create-subnet \
    --cidr-block 10.0.3.0/24 \
    --vpc-id vpc-xxxxxxxx \
    --outpost-arn arn:aws:outposts:us-west-2:123456789012:outpost/op-
xxxxxxxxxxxxxxxx \
    --availability-zone-id usw2-az1
```

5. Create a security group for the container instances, specifying the proper CIDR range for the AWS Outposts. (This step is different for AWS Outposts.)

```
aws ec2 create-security-group --group-name MyOutpostSG
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
    --port 22 --cidr 10.0.3.0/24
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
    --port 80 --cidr 10.0.3.0/24
```

6. Create the Cluster.

7. Define the Amazon ECS container agent environment variables to launch the instance into the cluster created in the previous step and define any tags you want to add to help identify the cluster (for example, `Outpost` to indicate that the cluster is for an Outpost).

```
#! /bin/bash
cat << ‘EOF’ >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_IMAGE_PULL_BEHAVIOR=prefer-cached
ECS_CONTAINER_INSTANCE_TAGS={“environment”: ”Outpost”}
EOF
```

> **Note**
>
> In order to avoid delays caused by pulling container images from Amazon ECR in the Region, use image caches. To do this, each time a task is run, configure the Amazon ECS agent to default to using the cached image on the instance itself by setting `ECS_IMAGE_PULL_BEHAVIOR` to `prefer-cached`.

8. Create the container instance, specifying the VPC and subnet for the AWS Outposts where this instance should run and an instance type that is available on the AWS Outposts. (This step is different for AWS Outposts.)

   The `userdata.txt` file contains the user data the instance can use to perform common automated configuration tasks and even run scripts after the instance starts. For information about the file for API calls, see Run commands on your Linux instance at launch in the *Amazon EC2 User Guide*.

   ```
   aws ec2 run-instances --count 1 --image-id ami-xxxxxxxx --instance-type c5.large \
       --key-name aws-outpost-key --subnet-id subnet-xxxxxxxxxxxxxxxxx \
       --iam-instance-profile Name outpost --security-group-id sg-xxxxxx \
       --associate-public-ip-address --user-data file://userdata.txt
   ```

   Note

   This command is also used when adding additional instances to the cluster. Any containers deployed in the cluster will be placed on that specific AWS Outposts.

Amazon ECS networking best practices

Modern applications are typically built out of multiple distributed components that communicate with each other. For example, a mobile or web application might communicate with an API endpoint, and the API might be powered by multiple microservices that communicate over the internet.

For information about the best practices for connection applications to the internet, see Connect Amazon ECS applications to the internet.

For information about the best practices for receiving inbound connections to Amazon ECS from the internet, see Best practices for receiving inbound connections to Amazon ECS from the internet.

For information about the best practices for connecting Amazon ECS to other AWS services, see Best practices for connecting Amazon ECS to AWS services from inside your VPC.

For information about the best practices for connecting services within a VPC, see Best practices for connecting Amazon ECS services in a VPC.

For information about the best practices for networking services across AWS accounts and VPCs, see Best practices for networking Amazon ECS services across AWS accounts and VPCs.

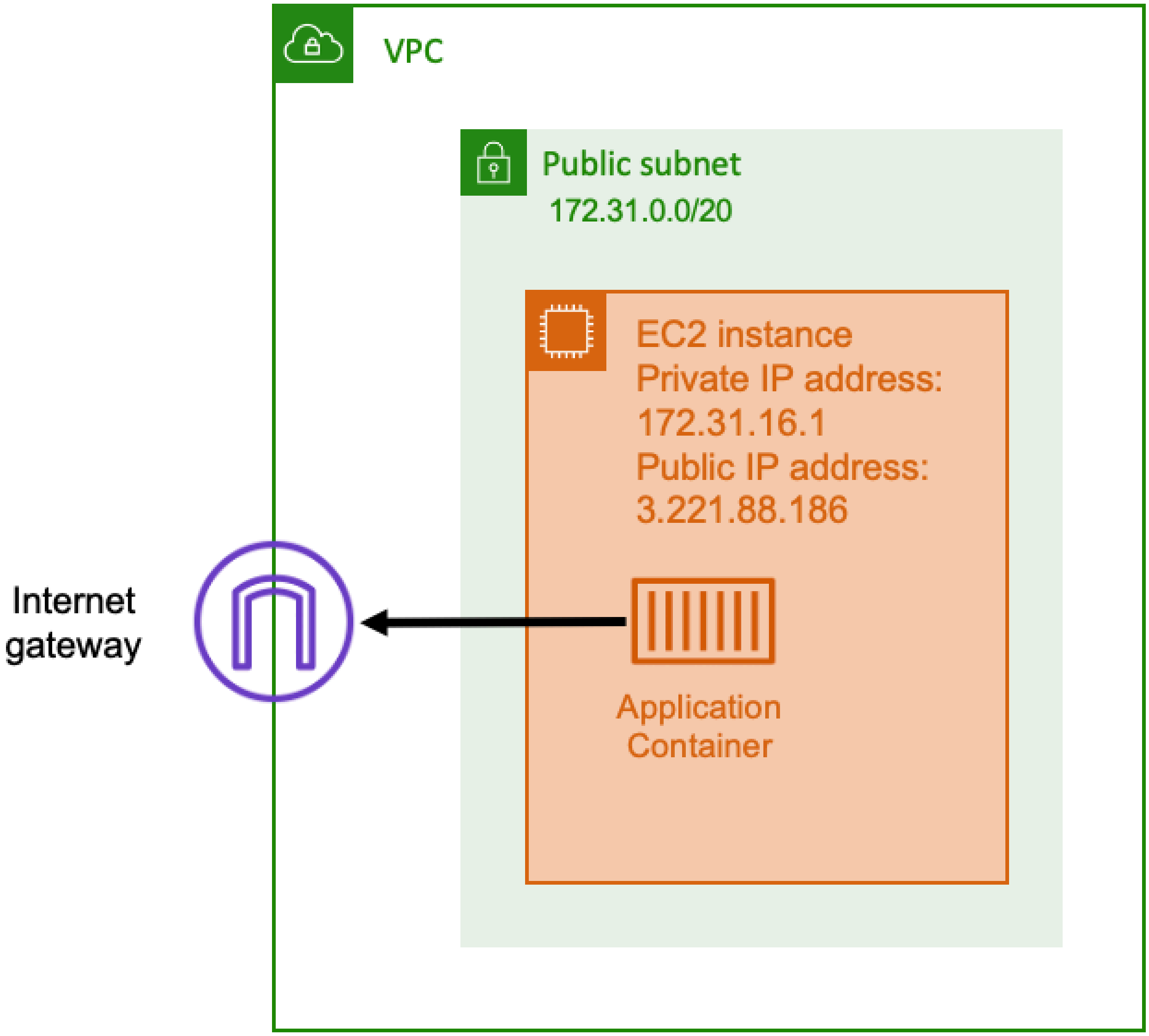
For information about the best practices for services to trobleshoot networking issues, see AWS services for Amazon ECS networking troubleshooting.

Connect Amazon ECS applications to the internet

Most containerized applications have a least some components that need outbound access to the internet. For example, the backend for a mobile app requires outbound access to push notifications.

Amazon Virtual Private Cloud has two main methods for facilitating communication between your VPC and the internet.

Public subnet and internet gateway



When you use a public subnet that has a route to an internet gateway, your containerized application can run on a host inside a VPC on a public subnet. The host that runs your container is assigned a public IP address. This public IP address is routable from the internet. For more information, see [Internet gateways](#) in the *Amazon VPC User Guide*.

This network architecture facilitates direct communication between the host that runs your application and other hosts on the internet. The communication is bi-directional. This means that not only can you establish an outbound connection to any other host on the internet, but other hosts on the internet might also attempt to connect to your host. Therefore, you should pay close attention to your security group and firewall rules. This ensures that other hosts on the internet can’t open any connections that you don't want to be opened.

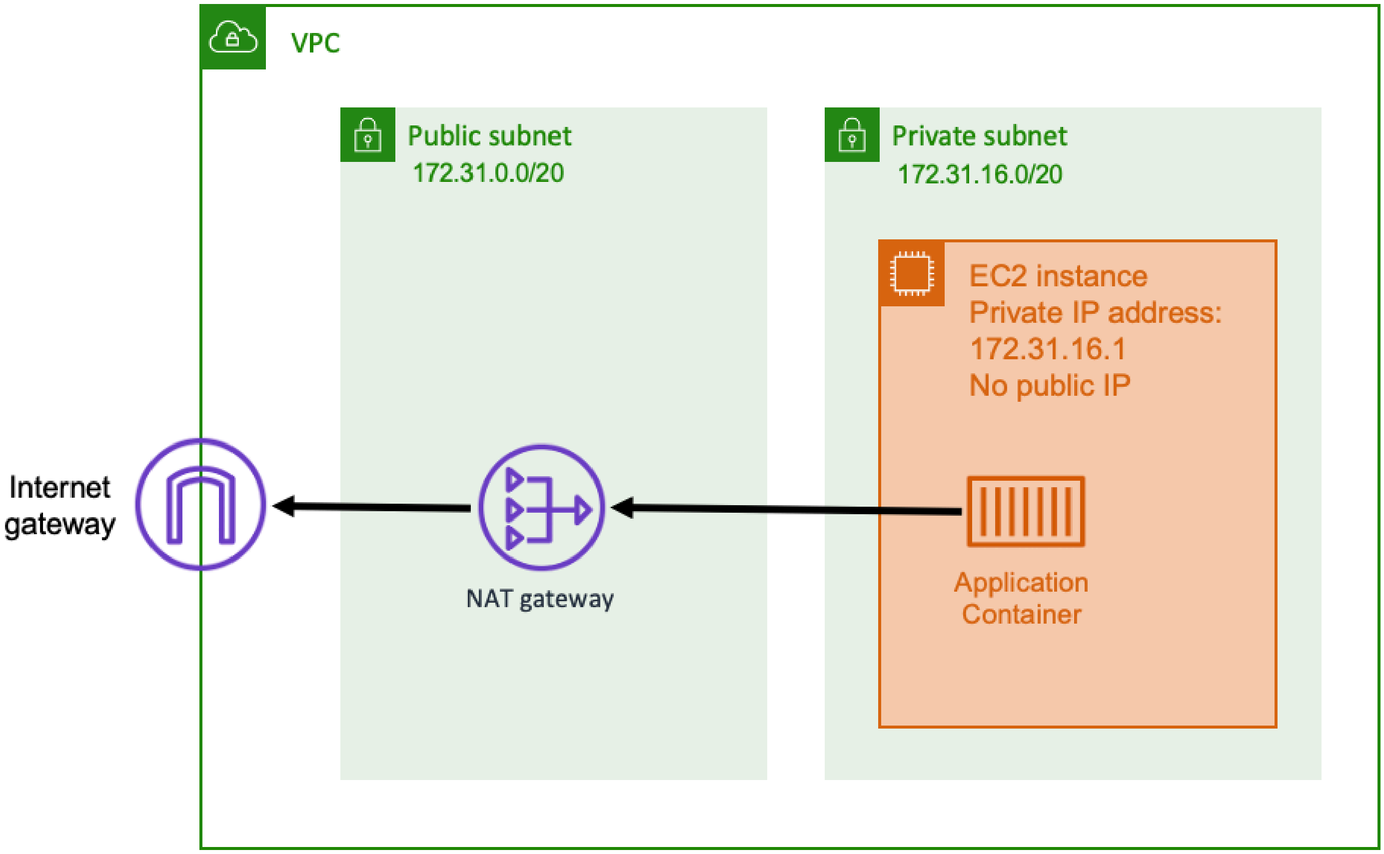
For example, if your application runs on Amazon EC2, make sure that port 22 for SSH access is not open. Otherwise, your instance could receive constant SSH connection attempts from malicious bots on the internet. These bots trawl through public IP addresses. After they find an open SSH port, they attempt to brute-force passwords to try to access your instance. Because of this, many organizations limit the usage of public subnets and prefer to have most, if not all, of their resources inside of private subnets.

Using public subnets for networking is suitable for public applications that require large amounts of bandwidth or minimal latency. Applicable use cases include video streaming and gaming services.

This networking approach is supported both when you use Amazon ECS on Amazon EC2 and when you use it on AWS Fargate.

- Amazon EC2 — You can launch EC2 instances on a public subnet. Amazon ECS uses these EC2 instances as cluster capacity, and any containers that are running on the instances can use the underlying public IP address of the host for outbound networking. This applies to both the `host` and `bridge` network modes. However, the `awsvpc` network mode doesn't provide task ENIs with public IP addresses. Therefore, they can’t make direct use of an internet gateway.
- Fargate — When you create your Amazon ECS service, specify public subnets for the networking configuration of your service, and use the **Assign public IP address** option. Each Fargate task is networked in the public subnet, and has its own public IP address for direct communication with the internet.

Private subnet and NAT gateway



When you use a private subnet and a NAT gateway, you can run your containerized application on a host that's in a private subnet. As such, this host has a private IP address that's routable inside your VPC, but isn't routable from the internet. This means that other hosts inside the VPC can connect to the host using its private IP address, but other hosts on the internet can't make any inbound communications to the host.

With a private subnet, you can use a Network Address Translation (NAT) gateway to allow a host inside a private subnet to connect to the internet. Hosts on the internet receive an inbound connection that appears to be coming from the public IP address of the NAT gateway that's inside a public subnet. The NAT gateway is responsible for serving as a bridge between the internet and the private subnet. This configuration is often preferred for security reasons because it means that your VPC is protected from direct access by attackers on the internet. For more information, see NAT gateways in the *Amazon VPC User Guide*.

This private networking approach is suitable for scenarios where you want to protect your containers from direct external access. Applicable scenarios include payment processing systems or containers storing user data and passwords. You're charged for creating and using a NAT gateway in your account. NAT gateway hourly usage and data processing rates also apply. For redundancy purposes, you should have a NAT gateway in each Availability Zone. This way, the loss in availability of a single Availability Zone doesn't compromise your outbound connectivity. Because of this, if you have a small workload, it might be more cost effective to use private subnets and NAT gateways.

This networking approach is supported both when using Amazon ECS on Amazon EC2 and when using it on AWS Fargate.

- Amazon EC2 — You can launch EC2 instances on a private subnet. The containers that run on these EC2 hosts use the underlying hosts networking, and outbound requests go through the NAT gateway.
- Fargate — When you create your Amazon ECS service, specify private subnets for the networking configuration of your service, and don't use the **Assign public IP address** option. Each Fargate task is hosted in a private subnet. Its outbound traffic is routed through any NAT gateway that you have associated with that private subnet.

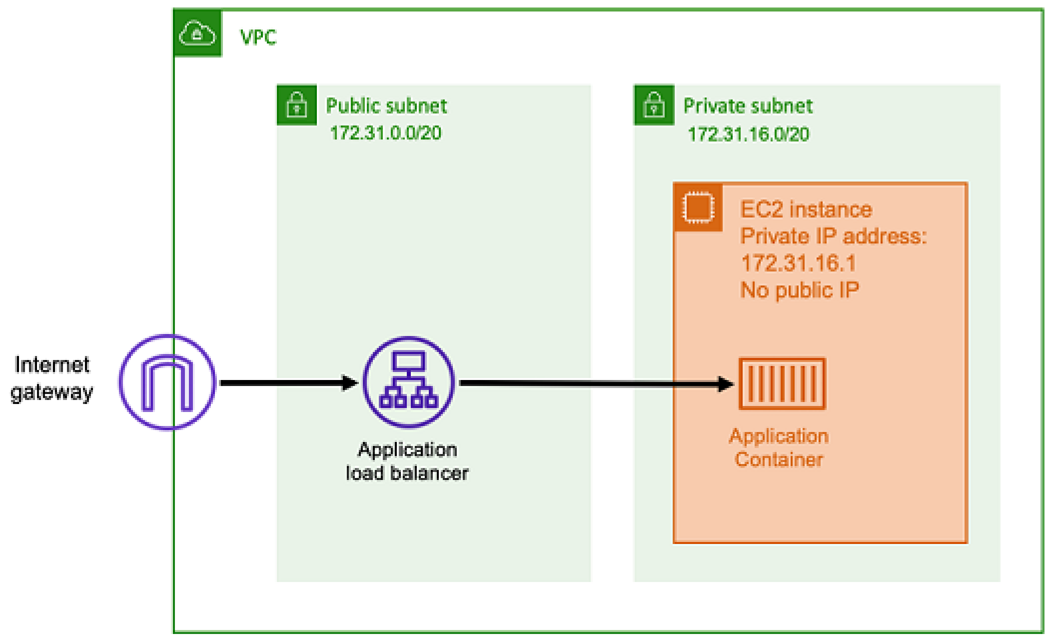
Best practices for receiving inbound connections to Amazon ECS from the internet

If you run a public service, you must accept inbound traffic from the internet. For example, your public website must accept inbound HTTP requests from browsers. In such case, other hosts on the internet must also initiate an inbound connection to the host of your application.

One approach to this problem is to launch your containers on hosts that are in a public subnet with a public IP address. However, we don't recommend this for large-scale applications. For these, a better approach is to have a scalable input layer that sits between the internet and your application. For this approach, you can use any of the AWS services listed in this section as an input.

Application Load Balancer

An Application Load Balancer functions at the application layer. It's the seventh layer of the Open Systems Interconnection (OSI) model. This makes an Application Load Balancer suitable for public HTTP services. If you have a website or an HTTP REST API, then an Application Load Balancer is a suitable load balancer for this workload. For more information, see What is an Application Load Balancer? in the *User Guide for Application Load Balancers*.



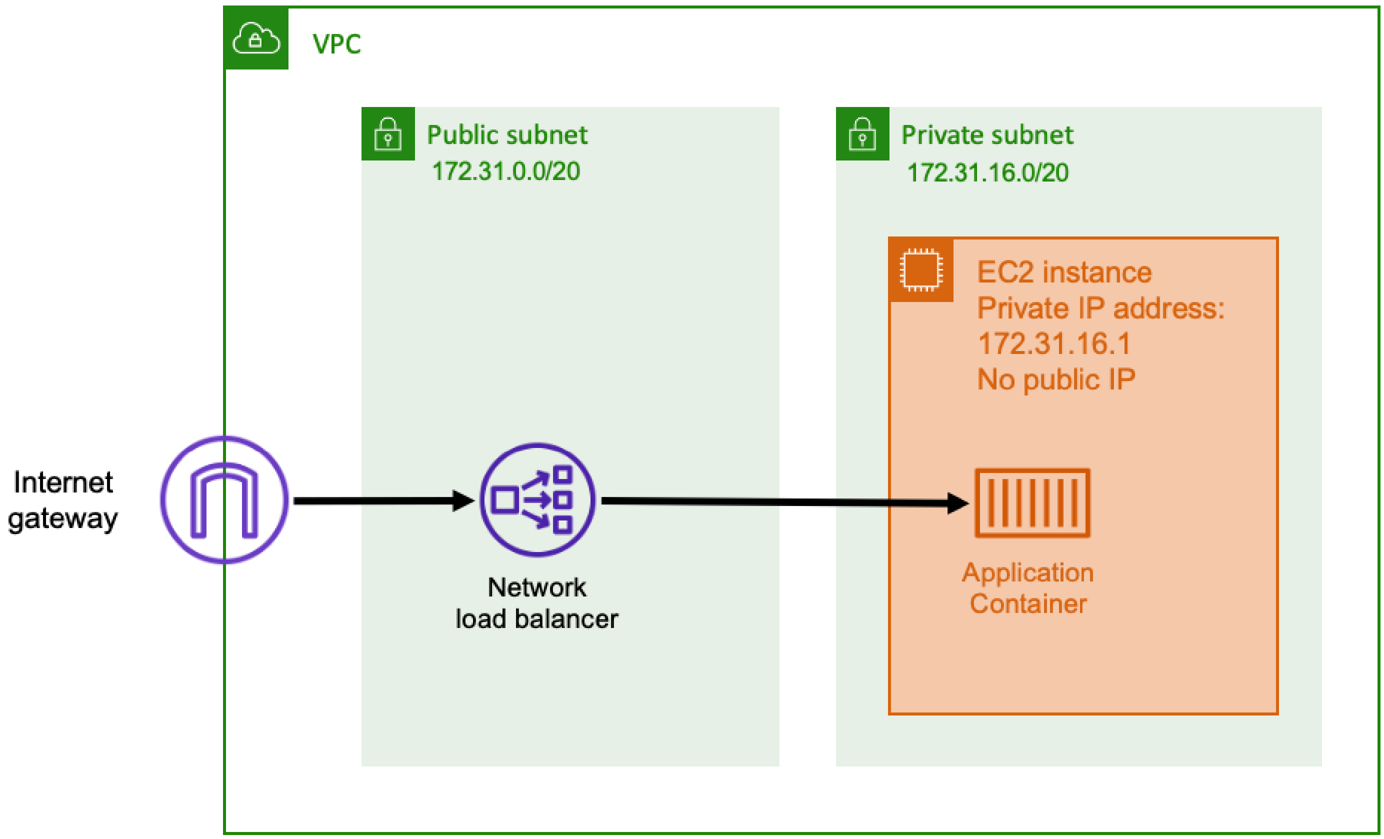
With this architecture, you create an Application Load Balancer in a public subnet so that it has a public IP address and can receive inbound connections from the internet. When the Application Load Balancer receives an inbound connection, or more specifically an HTTP request, it opens a connection to the application using its private IP address. Then, it forwards the request over the internal connection.

An Application Load Balancer has the following advantages.

- SSL/TLS termination — An Application Load Balancer can sustain secure HTTPS communication and certificates for communications with clients. It can optionally terminate the SSL connection at the load balancer level so that you don't have to handle certificates in your own application.
- Advanced routing — An Application Load Balancer can have multiple DNS hostnames. It also has advanced routing capabilities to send incoming HTTP requests to different destinations based on metrics such as the hostname or the path of the request. This means that you can use a single Application Load Balancer as the input for many different internal services, or even microservices on different paths of a REST API.
- gRPC support and websockets — An Application Load Balancer can handle more than just HTTP. It can also load balance gRPC and websocket based services, with HTTP/2 support.
- Security — An Application Load Balancer helps protect your application from malicious traffic. It includes features such as HTTP de sync mitigations, and is integrated with AWS Web Application Firewall (AWS WAF). AWS WAF can further filter out malicious traffic that might contain attack patterns, such as SQL injection or cross-site scripting.

Network Load Balancer

A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection (OSI) model. It's suitable for non-HTTP protocols or scenarios where end-to-end encryption is necessary, but doesn’t have the same HTTP-specific features of an Application Load Balancer. Therefore, a Network Load Balancer is best suited for applications that don’t use HTTP. For more information, see [What is a Network Load Balancer?](#) in the *User Guide for Network Load Balancers*.



When a Network Load Balancer is used as an input, it functions similarly to an Application Load Balancer. This is because it's created in a public subnet and has a public IP address that can be accessed on the internet. The Network Load Balancer then opens a connection to the private IP address of the host running your container, and sends the packets from the public side to the private side.

Network Load Balancer features

Because the Network Load Balancer operates at a lower level of the networking stack, it doesn't have the same set of features that Application Load Balancer does. However, it does have the following important features.

- End-to-end encryption — Because a Network Load Balancer operates at the fourth layer of the OSI model, it doesn't read the contents of packets. This makes it suitable for load balancing communications that need end-to-end encryption.
- TLS encryption — In addition to end-to-end encryption, Network Load Balancer can also terminate TLS connections. This way, your backend applications don't have to implement their own TLS.
- UDP support — Because a Network Load Balancer operates at the fourth layer of the OSI model, it's suitable for non HTTP workloads and protocols other than TCP.

Closing connections

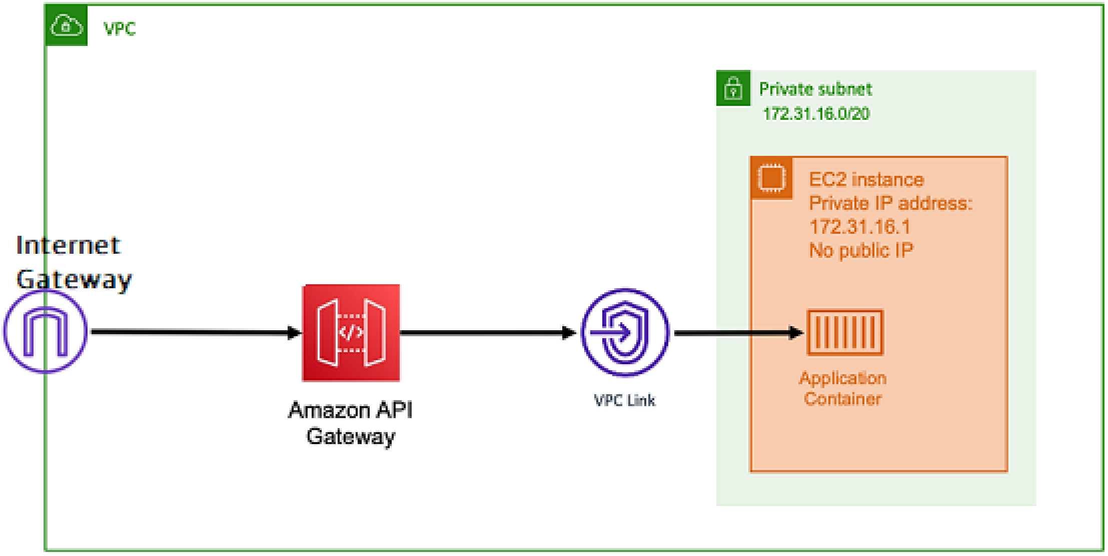
Because the Network Load Balancer does not observe the application protocol at the higher layers of the OSI model, it cannot send closure messages to the clients in those protocols. Unlike the Application Load Balancer, those connections need to be closed by the application or you can configure the Network Load Balancer to close the fourth layer connections when a task is stopped or replaced. See the connection termination setting for Network Load Balancer target groups in the Network Load Balancer documentation.

Letting the Network Load Balancer close connections at the fourth layer can cause clients to display undesired error messages, if the client does not handle them. See the Builders Library for more information on recommended client configuration here.

The methods to close connections will vary by application, however one way is to ensure that the Network Load Balancer target deregistration delay is longer than client connection timeout. The client would timeout first and reconnect gracefully through the Network Load Balancer to the next task while the old task slowly drains all of its clients. For more information about the Network Load Balancer target deregistration delay, see the Network Load Balancer documentation.

Amazon API Gateway HTTP API

Amazon API Gateway is suitable for HTTP applications with sudden bursts in request volumes or low request volumes. For more information, see What is Amazon API Gateway? in the *API Gateway Developer Guide*.



The pricing model for both Application Load Balancer and Network Load Balancer include an hourly price to keep the load balancers available for accepting incoming connections at all times. In contrast, API Gateway charges for each request separately. This has the effect that, if no requests come in, there are no charges. Under high traffic loads, an Application Load Balancer or Network Load Balancer can handle a greater volume of requests at a cheaper per-request price than API Gateway. However, if you have a low number of requests overall or have periods of low traffic, then the cumulative price for using the API Gateway should be more cost effective than paying a hourly charge to maintain a load balancer that's being underutilized. The API Gateway can also cache API responses, which might result in lower backend request rates.

API Gateway functions which use a VPC link that allows the AWS managed service to connect to hosts inside the private subnet of your VPC, using its private IP address. It can detect these private IP addresses by looking at AWS Cloud Map service discovery records that are managed by Amazon ECS Service Discovery.

API Gateway supports the following features.

- The API Gateway operation is similar to a load balancer, but has additional capabilities unique to API management
- The API Gateway provides additional capabilities around client authorization, usage tiers, and request/response modification. For more information, see Amazon API Gateway features.
- The API Gateway can support edge, regional, and private API gateway endpoints. Edge endpoints are available through a managed CloudFront distribution. Regional and private endpoints are both local to a Region.

- SSL/TLS termination
- Routing different HTTP paths to different backend microservices

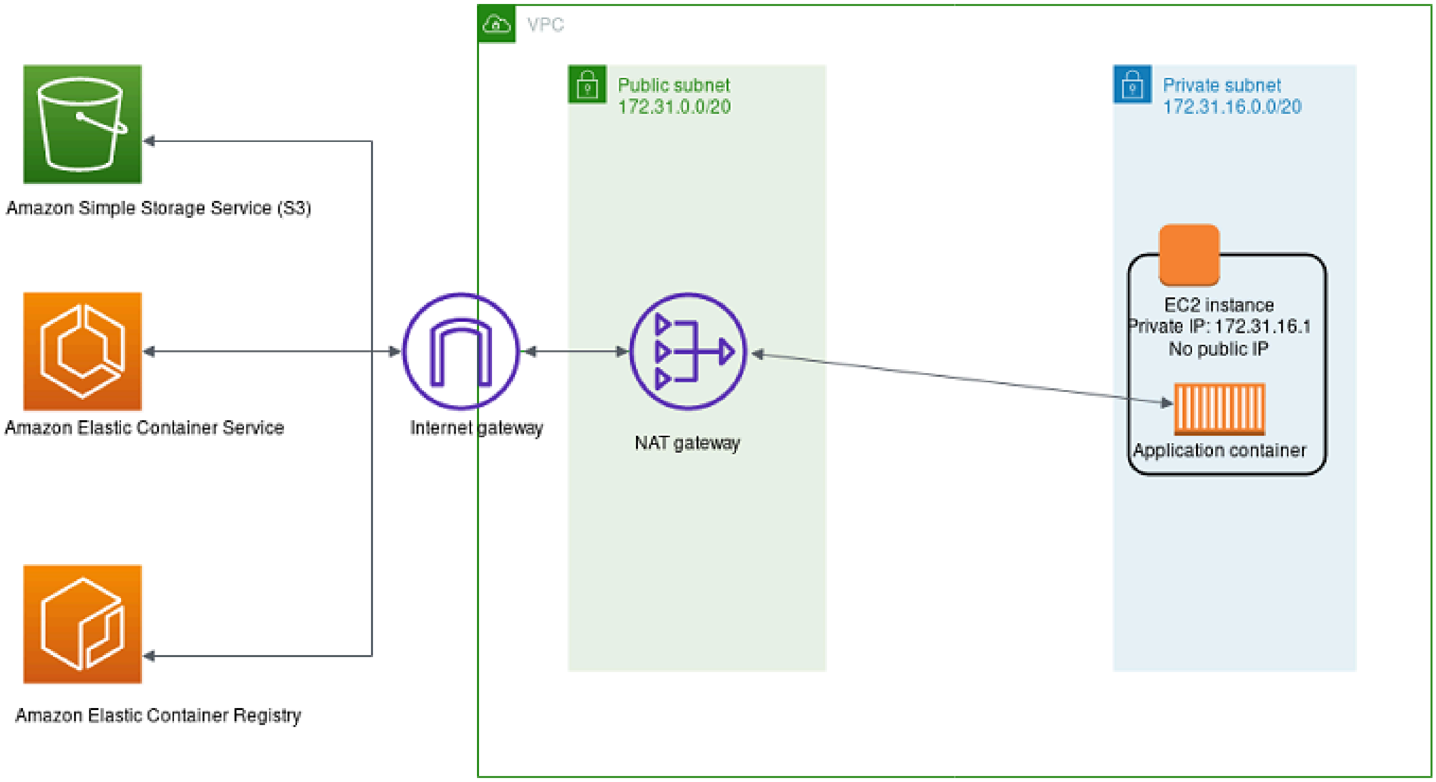
Besides the preceding features, API Gateway also supports using custom Lambda authorizers that you can use to protect your API from unauthorized usage. For more information, see Field Notes: Serverless Container-based APIs with Amazon ECS and Amazon API Gateway.

Best practices for connecting Amazon ECS to AWS services from inside your VPC

For Amazon ECS to function properly, the Amazon ECS container agent that runs on each host must communicate with the Amazon ECS control plane. If you're storing your container images in Amazon ECR, the Amazon EC2 hosts must communicate to the Amazon ECR service endpoint, and to Amazon S3, where the image layers are stored. If you use other AWS services for your containerized application, such as persisting data stored in DynamoDB, double-check that these services also have the necessary networking support.

NAT gateway

Using a NAT gateway is the easiest way to ensure that your Amazon ECS tasks can access other AWS services. For more information about this approach, see Private subnet and NAT gateway.



The following are the disadvantages to using this approach:

- You can't limit what destinations the NAT gateway can communicate with. You also can't limit which destinations your backend tier can communicate to without disrupting all outbound communications from your VPC.
- NAT gateways charge for every GB of data that passes through. If you use the NAT gateway for any of the following operations, you're charged for every GB of bandwidth:
  - Downloading large files from Amazon S3
  - Doing a high volume of database queries to DynamoDB
  - Pulling images from Amazon ECR

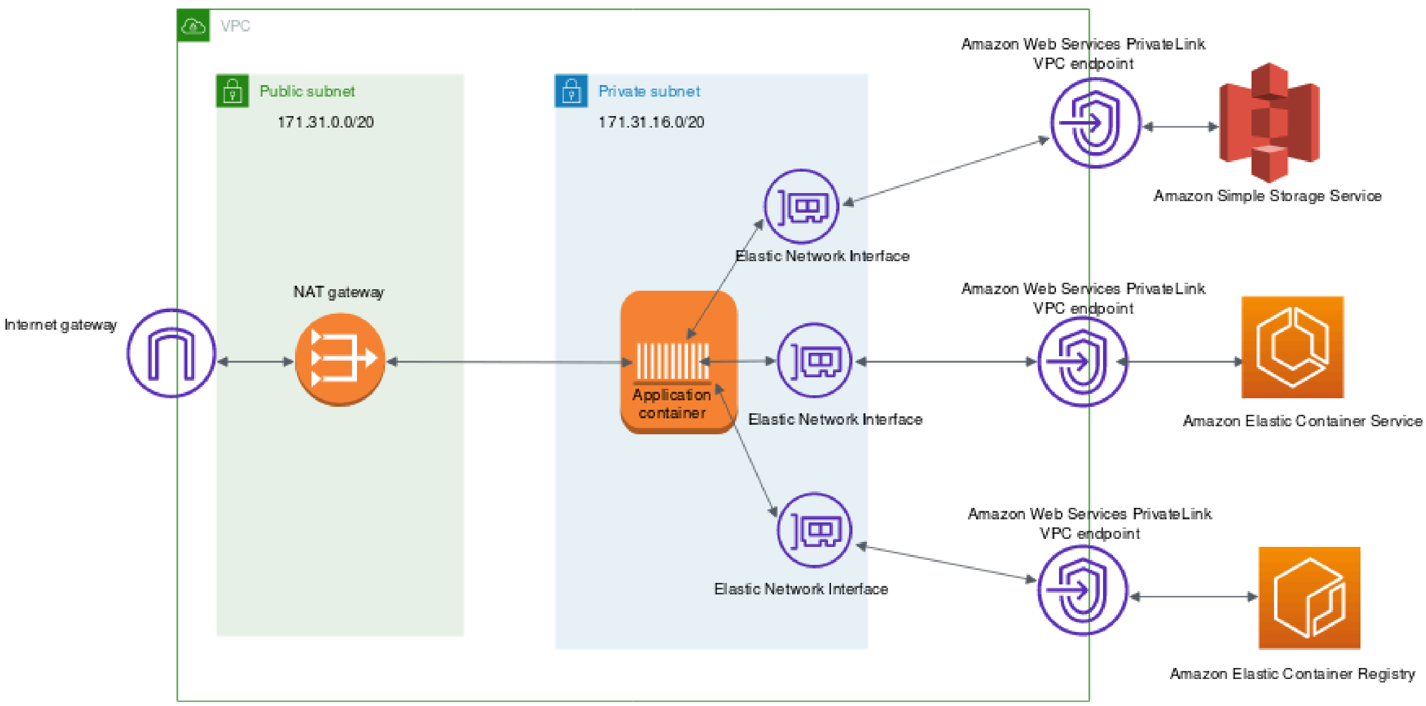
  Additionally, NAT gateways support 5 Gbps of bandwidth and automatically scale up to 45 Gbps. If you route through a single NAT gateway, applications that require very high bandwidth connections might encounter networking constraints. As a workaround, you can divide your workload across multiple subnets and give each subnet its own NAT gateway.

AWS PrivateLink

AWS PrivateLink provides private connectivity between VPCs, AWS services, and your on-premises networks without exposing your traffic to the public internet.

A VPC endpoint allows private connections between your VPC and supported AWS services and VPC endpoint services. Traffic between your VPC and the other service doesn't leave the Amazon network. A VPC endpoint doesn't require an internet gateway, virtual private gateway, NAT device, VPN connection, or AWS Direct Connect connection. Amazon EC2 instances in your VPC don't require public IP addresses to communicate with resources in the service.

The following diagram shows how communication to AWS services works when you are using VPC endpoints instead of an internet gateway. AWS PrivateLink provisions elastic network interfaces (ENIs) inside of the subnet, and VPC routing rules are used to send any communication to the service hostname through the ENI, directly to the destination AWS service. This traffic no longer needs to use the NAT gateway or internet gateway.



The following are some of the common VPC endpoints that are used with the Amazon ECS service.

- Gateway endpoints for Amazon S3
- DynamoDB VPC endpoint
- Amazon ECS VPC endpoint
- Amazon ECR VPC endpoint

Many other AWS services support VPC endpoints. If you make heavy usage of any AWS service, you should look up the specific documentation for that service and how to create a VPC endpoint for that traffic.

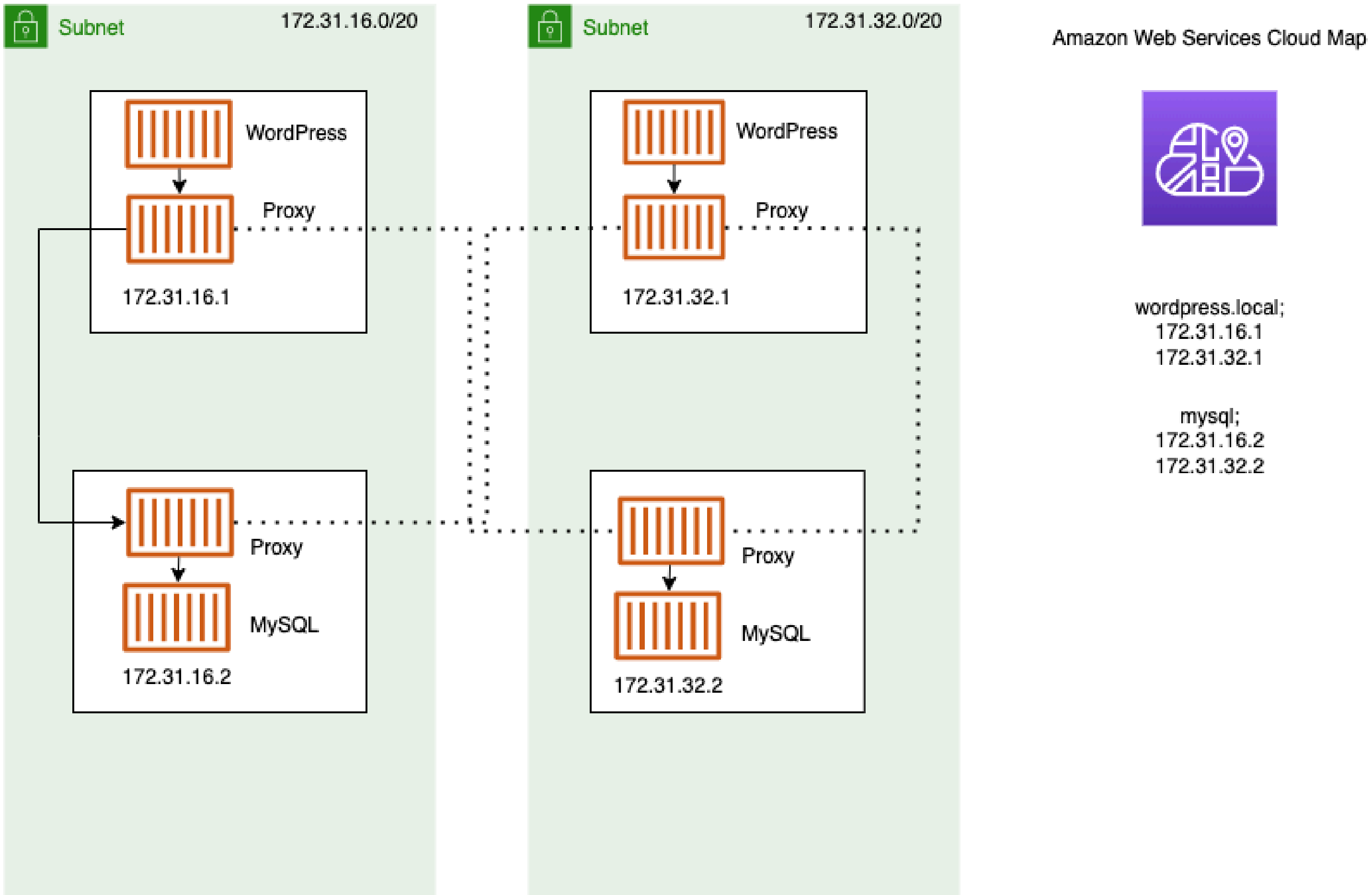
Best practices for connecting Amazon ECS services in a VPC

Using Amazon ECS tasks in a VPC, you can split monolithic applications into separate parts that can be deployed and scaled independently in a secure environment. This architecture is called service-oriented architecture (SOA) or microservices. However, it can be challenging to make sure that all of these parts, both in and outside of a VPC, can communicate with each other. There are several approaches for facilitating communication, all with different advantages and disadvantages.

Using Service Connect

We recommend Service Connect, which provides Amazon ECS configuration for service discovery, connectivity, and traffic monitoring. With Service Connect, your applications can use short names and standard ports to connect to services in the same cluster, other clusters, including across VPCs in the same Region. For more information, see Amazon ECS Service Connect.

When you use Service Connect, Amazon ECS manages all of the parts of service discovery: creating the names that can be discovered, dynamically managing entries for each task as the tasks start and stop, running an agent in each task that is configured to discover the names. Your application can look up the names by using the standard functionality for DNS names and making connections. If your application does this already, you don't need to modify your application to use Service Connect.

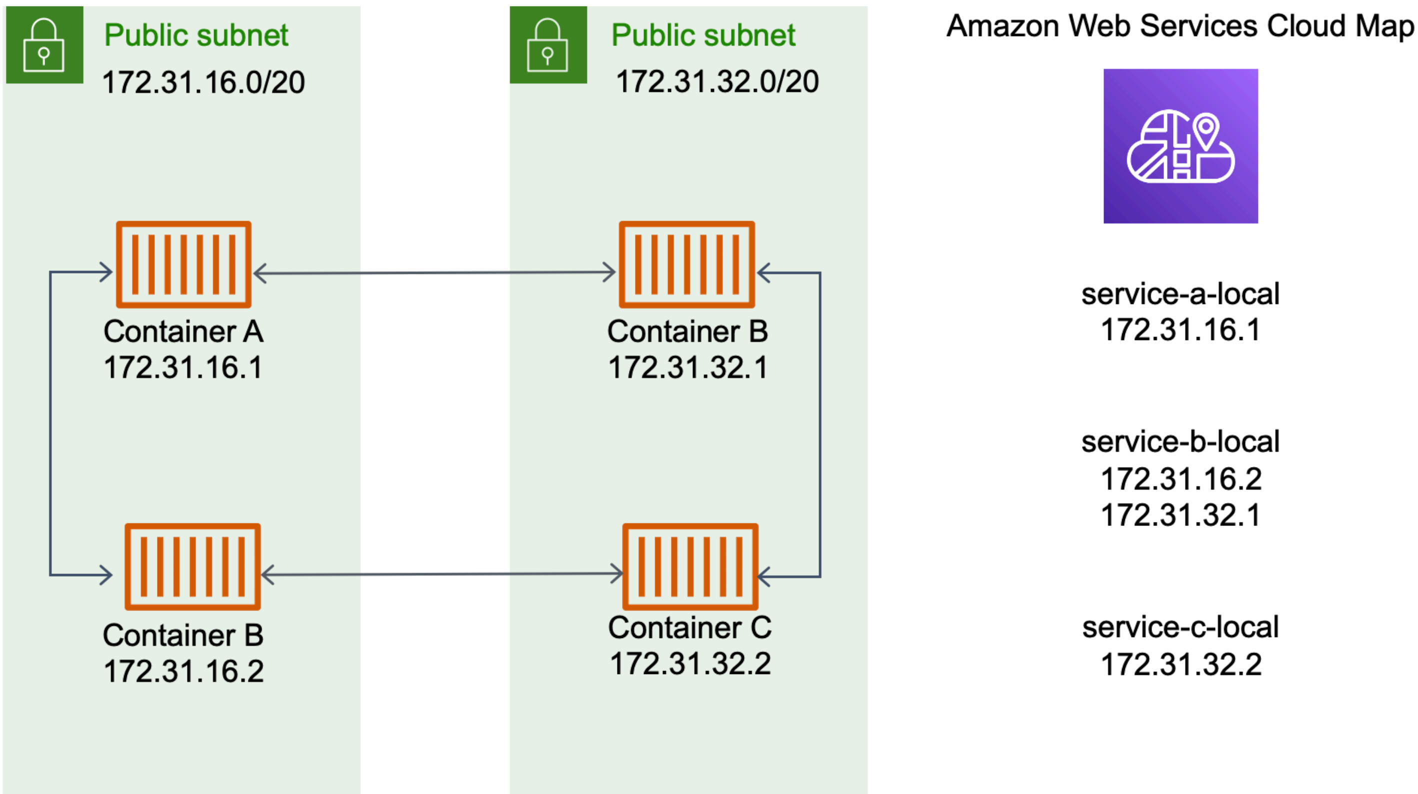


Changes only happen during deployments

You provide the complete configuration inside each service and task definition. Amazon ECS manages changes to this configuration in each service deployment, to ensure that all tasks in a deployment behave in the same way. For example, a common problem with DNS as service discovery is controlling a migration. If you change a DNS name to point to the new replacement IP addresses, it might take the maximum TTL time before all the clients begin using the new service. With Service Connect, the client deployment updates the configuration by replacing the client tasks. You can configure the deployment circuit breaker and other deployment configuration to affect Service Connect changes in the same way as any other deployment.

Using service discovery

Another approach for service-to-service communication is direct communication using service discovery. In this approach, you can use the AWS Cloud Map service discovery integration with Amazon ECS. Using service discovery, Amazon ECS syncs the list of launched tasks to AWS Cloud Map, which maintains a DNS hostname that resolves to the internal IP addresses of one or more tasks from that particular service. Other services in the Amazon VPC can use this DNS hostname to send traffic directly to another container using its internal IP address. For more information, see Service discovery.



In the preceding diagram, there are three services. `service-a-local` has one container and communicates with `service-b-local`, which has two containers. `service-b-local` must also communicate with `service-c-local`, which has one container. Each container in all three of these services can use the internal DNS names from AWS Cloud Map to find the internal IP addresses of a container from the downstream service that it needs to communicate to.

This approach to service-to-service communication provides low latency. At first glance, it's also simple as there are no extra components between the containers. Traffic travels directly from one container to the other container.

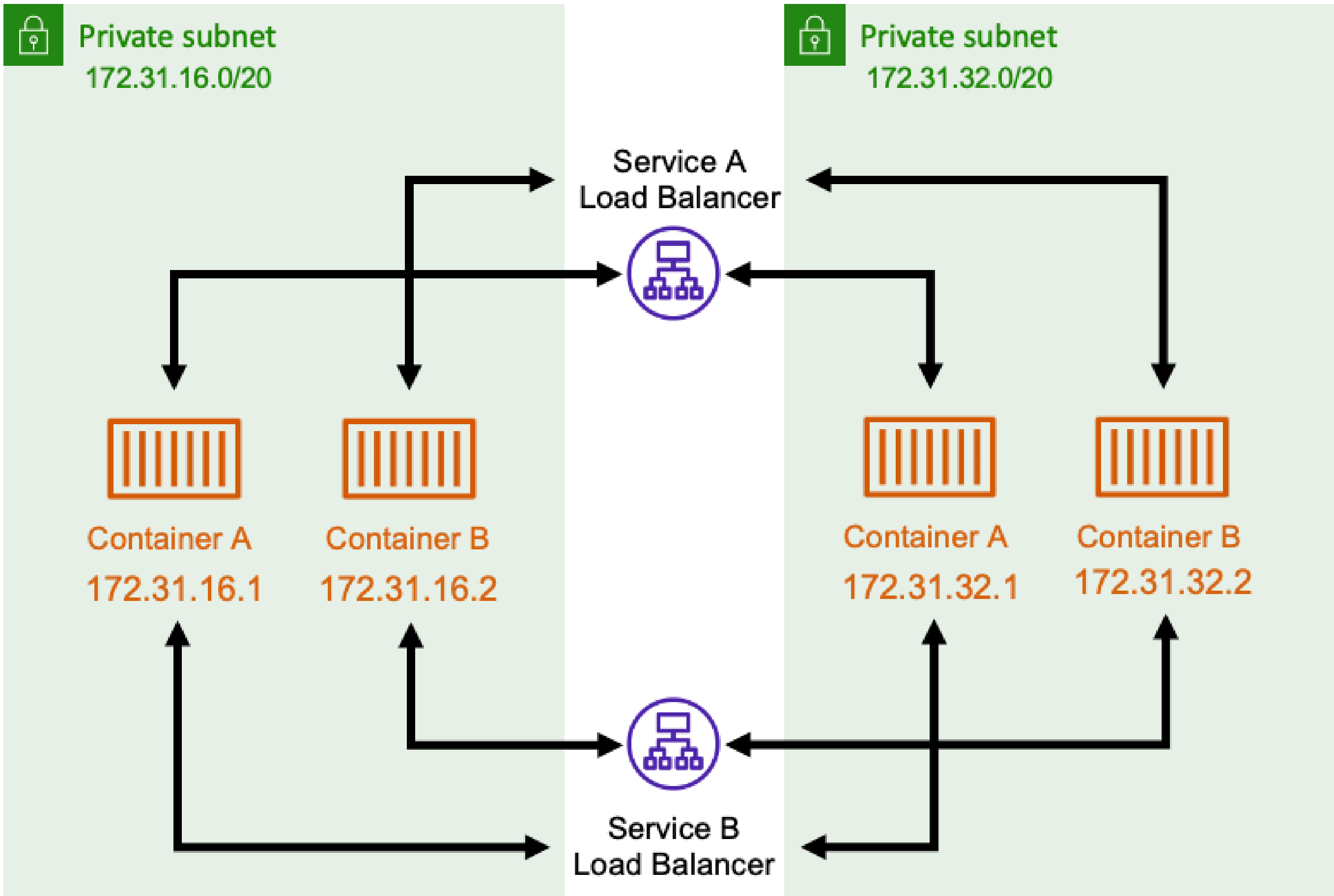
This approach is suitable when using the `awsvpc` network mode, where each task has its own unique IP address. Most software only supports the use of DNS `A` records, which resolve directly to IP addresses. When using the `awsvpc` network mode, the IP address for each task are an `A` record. However, if you're using `bridge` network mode, multiple containers could be sharing the same IP address. Additionally, dynamic port mappings cause the containers to be randomly assigned port numbers on that single IP address. At this point, an `A` record is no longer be enough for service discovery. You must also use an SRV record. This type of record can keep track of both IP addresses and port numbers but requires that you configure applications appropriately. Some prebuilt applications that you use might not support SRV records.

Another advantage of the `awsvpc` network mode is that you have a unique security group for each service. You can configure this security group to allow incoming connections from only the specific upstream services that need to talk to that service.

The main disadvantage of direct service-to-service communication using service discovery is that you must implement extra logic to have retries and deal with connection failures. DNS records have a time-to-live (TTL) period that controls how long they are cached for. It takes some time for the DNS record to be updated and for the cache to expire so that your applications can pick up the latest version of the DNS record. So, your application might end up resolving the DNS record to point at another container that's no longer there. Your application needs to handle retries and have logic to ignore bad backends.

Using an internal load balancer

Another approach to service-to-service communication is to use an internal load balancer. An internal load balancer exists entirely inside of your VPC and is only accessible to services inside of your VPC.



The load balancer maintains high availability by deploying redundant resources into each subnet. When a container from `serviceA` needs to communicate with a container from `serviceB`, it opens a connection to the load balancer. The load balancer then opens a connection to a container from `service B`. The load balancer serves as a centralized place for managing all connections between each service.

If a container from `serviceB` stops, then the load balancer can remove that container from the pool. The load balancer also does health checks against each downstream target in its pool and can automatically remove bad targets from the pool until they become healthy again. The applications no longer need to be aware of how many downstream containers there are. They just open their connections to the load balancer.

This approach is advantageous to all network modes. The load balancer can keep track of task IP addresses when using the `awsvpc` network mode, as well as more advanced combinations of IP addresses and ports when using the `bridge` network mode. It evenly distributes traffic across all the IP address and port combinations, even if several containers are actually hosted on the same Amazon EC2 instance, just on different ports.

The one disadvantage of this approach is cost. To be highly available, the load balancer needs to have resources in each Availability Zone. This adds extra cost because of the overhead of paying for the load balancer and for the amount of traffic that goes through the load balancer.

However, you can reduce overhead costs by having multiple services share a load balancer. This is particularly suitable for REST services that use an Application Load Balancer. You can create path-based routing rules that route traffic to different services. For example, `/api/user/*` might route to a container that's part of the `user` service, whereas `/api/order/*` might route to the associated `order` service. With this approach, you only pay for one Application Load Balancer, and have one consistent URL for your API. However, you can split the traffic off to various microservices on the backend.

Best practices for networking Amazon ECS services across AWS accounts and VPCs

If you're part of an organization with multiple teams and divisions, you probably deploy services independently into separate VPCs inside a shared AWS account or into VPCs that are associated with multiple individual AWS accounts. No matter which way you deploy your services, we recommend that you supplement your networking components to help route traffic between VPCs. For this, several AWS services can be used to supplement your existing networking components.

- AWS Transit Gateway — You should consider this networking service first. This service serves as a central hub for routing your connections between Amazon VPCs, AWS accounts, and on-premises networks. For more information, see [What is a transit gateway?](#) in the *Amazon VPC Transit Gateways Guide*.
- Amazon VPC and VPN support — You can use this service to create site-to-site VPN connections for connecting on-premises networks to your VPC. For more information, see [What is AWS Site-to-Site VPN?](#) in the *AWS Site-to-Site VPN User Guide*.
- Amazon VPC — You can use Amazon VPC peering to help you to connect multiple VPCs, either in the same account, or across accounts. For more information, see [What is VPC peering?](#) in the *Amazon VPC Peering Guide*.
- Shared VPCs — You can use a VPC and VPC subnets across multiple AWS accounts. For more information, see [Working with shared VPCs](#) in the *Amazon VPC User Guide*.

AWS services for Amazon ECS networking troubleshooting

The following services and features can help you to gain insights about your network and service configurations. You can use this information to troubleshoot networking issues and better optimize your services.

CloudWatch Container Insights

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. Metrics include the utilization of resources such as CPU, memory, disk, and network. They're available in CloudWatch automatic dashboards. For more information, see Setting up Container Insights on Amazon ECS in the *Amazon CloudWatch User Guide*.

AWS X-Ray

AWS X-Ray is a tracing service that you can use to collect information about the network requests that your application makes. You can use the SDK to instrument your application and capture timings and response codes of traffic between your services, and between your services and AWS service endpoints. For more information, see What is AWS X-Ray in the *AWS X-Ray Developer Guide*.

You can also explore AWS X-Ray graphs of how your services network with each other. Or, use them to explore aggregate statistics about how each service-to-service link is performing. Last, you can dive deeper into any specific transaction to see how segments representing network calls are associated with that particular transaction.

You can use these features to identify if there's a networking bottleneck or if a specific service within your network isn't performing as expected.

VPC Flow Logs

You can use Amazon VPC flow logs to analyze network performance and debug connectivity issues. With VPC flow logs enabled, you can capture a log of all the connections in your VPC. These include connections to networking interfaces that are associated with Elastic Load Balancing, Amazon RDS, NAT gateways, and other key AWS services that you might be using. For more information, see VPC Flow Logs in the *Amazon VPC User Guide*.

Network tuning tips

There are a few settings that you can fine tune in order to improve your networking.

nofile ulimit

If you expect your application to have high traffic and handle many concurrent connections, you should take into account the system quota for the number of files allowed. When there are a lot of network sockets open, each one must be represented by a file descriptor. If your file descriptor quota is too low, it will limit your network sockets. This results in failed connections or errors. You can update the container specific quota for the number of files in the Amazon ECS task definition. If you're running on Amazon EC2 (instead of AWS Fargate), then you might also need to adjust these quotas on your underlying Amazon EC2 instance.

sysctl net

Another category of tunable setting is the `sysctl` net settings. You should refer to the specific settings for your Linux distribution of choice. Many of these settings adjust the size of read and write buffers. This can help in some situations when running large-scale Amazon EC2 instances that have a lot of containers on them.

Optimize Amazon ECS task launch time

In order to speed up your task launches, consider the following recommendations.

- **Cache container images and binpack instances**

  If you use the EC2 launch type you can configure the Amazon ECS container agent pull behavior to `ECS_IMAGE_PULL_BEHAVIOR`: `prefer-cached`. The image is pulled remotely if there is no cached image. Otherwise, the cached image on the instance is used. Automated image cleanup is turned off for the container to ensure that the cached image isn't removed. This reduces image pull-time for subsequent launches. The effect of caching is even greater when you have a high task density in your container instances, which you can configure using the `binpack` placement strategy. Caching container images is especially beneficial for windows-based workloads which usually have large (tens of GBs) container image sizes. When using the `binpack` placement strategy, you can also consider using Elastic Network Interface (ENI) trunking to place more tasks with the `awsvpc` network mode on each container instance. ENI trunking increases the number of tasks you can run on `awsvpc` mode. For example, a c5.large instance that may support running only 2 tasks concurrently, can run up to 10 tasks with ENI trunking.

- **Choose an optimal network mode**

  Although there are many instances where `awsvpc` network mode is ideal, this network mode can inherently increase task launch latency, because for each task in `awsvpc` mode, Amazon

ECS workflows need to provision and attach an ENI by invoking Amazon EC2 APIs which adds an overhead of several seconds to your task launches. By contrast, a key advantage of using `awsvpc` network mode is that each task has a security group to allow or deny traffic. This means you have greater flexibility to control communications between tasks and services at a more granular level. If the deployment speed is your priority, you can consider using `bridge` mode to speed up task launches. For more information, see the section called “AWSVPC network mode”.

- **Track your task launch lifecycle to find optimization opportunities**

  It is often difficult to know the amount of time it takes for your application to start-up. Launching your container image, running start-up scripts, and other configurations during application start-up can take a surprising amount of time. You can use the Task metadata endpoint to post metrics to track application start-up time from `StartedAt` in the container metadata response to the `StartedAt` time for your task or service. With this data, you can understand how your application is contributing to the total launch time, and find areas where you can reduce unnecessary application-specific overhead and optimize your container images. For more information, see Amazon ECS Auto scaling and capacity management best practices.

- **Choose an optimal instance type (for the EC2 launch type)**

  Choosing the correct instance type is based on the resource reservation (for example, CPU, memory) that you configure on your task. Therefore, when sizing the instance, you can calculate how many tasks can be placed on a single instance. A simple example of a well-placed task, is hosting 4 tasks requiring 0.5 vCPU and 2GB of memory reservations in an m5.large instance (supporting 2 vCPU and 8 GB memory). The reservations of this task definition take full advantage of the instance’s resources.

Operating Amazon ECS at scale

As you begin operating Amazon ECS at scale, consider how service quotas and API throttles for Amazon ECS and the AWS services that integrate with Amazon ECS might affect you.

Topics

Amazon ECS service quotas and API throttling limits

Amazon ECS integrates with several AWS services, including Elastic Load Balancing, AWS Cloud Map, and Amazon EC2. With this tight integration, Amazon ECS includes several features such as service load balancing, service discovery, task networking, and cluster auto scaling. Amazon ECS and the other AWS services that it integrates with all maintain service quotas and API rate limits to ensure consistent performance and utilization. These service quotas also prevent the accidental provisioning of more resources than needed and protect against malicious actions that might increase your bill.

By familiarizing yourself with your service quotas and the AWS API rate limits, you can plan for scaling your workloads without worrying about unexpected performance degradation. For more information, see Amazon ECS service quotas and Request throttling for the Amazon ECS API.

When scaling your workloads on Amazon ECS, consider the following service quota. For instructions on how to request a service quota increase, see Managing your Amazon ECS and AWS Fargate service quotas in the AWS Management Console.

- AWS Fargate has quotas that limit the number of concurrent running tasks in each AWS Region. There are quotas for both On-Demand and Fargate Spot tasks on Amazon ECS. Each service quota also includes any Amazon EKS pods that you run on Fargate. For more information about the Fargate quotas, see AWS Fargate service quotas in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- For tasks that run on Amazon EC2 instances, the maximum number of Amazon EC2 instances that you can register for each cluster is 5,000. If you use Amazon ECS cluster auto scaling with an Auto Scaling group capacity provider, or if you manage Amazon EC2 instances for your cluster on your own, this quota might become a deployment bottleneck. If you require more capacity, you can create more clusters or request a service quota increase.
- If you use Amazon ECS cluster auto scaling with an Auto Scaling group capacity provider, when scaling your services consider the `Tasks in the PROVISIONING state per cluster` quota. This quota is the maximum number of tasks in the `PROVISIONING` state for each cluster for which capacity providers can increase capacity. When you launch a large number of tasks all at the same time, you can easily meet this quota. One example is if you simultaneously deploy tens of services, each with hundreds of tasks. When this happens, the capacity provider needs to launch new container instances to place the tasks when the cluster has insufficient capacity. While the capacity provider is launching additional Amazon EC2 instances, the Amazon ECS service scheduler likely will continue to launch tasks in parallel. However, this activity might be

throttled because of insufficient cluster capacity. The Amazon ECS service scheduler implements a back-off and exponential throttling strategy for retrying task placement as new container instances are launched. As a result, you might experience slower deployment or scale-out times. To avoid this situation, you can plan your service deployments in one of the following ways. Either deploy a large number of tasks that don't require increasing cluster capacity, or keep spare cluster capacity for new task launches.

In addition to considering Amazon ECS service quota when scaling your workloads, consider also the service quota for the other AWS services that are integrated with Amazon ECS. The following section covers the key rate limits for each service in detail, and provides recommendations to deal with potential throttling issues.

Elastic Load Balancing

You can configure your Amazon ECS services to use Elastic Load Balancing to distribute traffic evenly across the tasks. For more information about how to choose a load balancer, see Use load balancing to distribute Amazon ECS service traffic.

Elastic Load Balancing service quotas

When you scale your workloads, consider the following Elastic Load Balancing service quotas. Most Elastic Load Balancing service quotas are adjustable, and you can request an increase in the Service Quotas console.

Application Load Balancer

When you use an Application Load Balancer, depending on your use case, you might need to request a quota increase for:

- The `Targets per Application Load Balancer` quota which is the number of targets behind your Application Load Balancer.
- The `Targets per Target Group per Region` quota which is the number of targets behind your Target Groups.

For more information, see Quotas for your Application Load Balancers.

Network Load Balancer

There are stricter limitations on the number of targets you can register with a Network Load Balancer. When using a Network Load Balancer, you often will want to enable cross-zone support, which comes with additional scaling limitations on `Targets per Availability Zone Per Network Load Balancer` the maximum number of targets per Availability Zone for each Network Load Balancer. For more information, see Quotas for your Network Load Balancers.

Elastic Load Balancing API throttling

When you configure an Amazon ECS service to use a load balancer, the target group health checks must pass before the service is considered healthy. For performing these health checks, Amazon ECS invokes Elastic Load Balancing API operations on your behalf. If you have a large number of services configured with load balancers in your account, you might experience slower service deployments because of potential throttling specifically for the `RegisterTarget`, `DeregisterTarget`, and `DescribeTargetHealth` Elastic Load Balancing API operations. When throttling occurs, throttling errors occur in your Amazon ECS service event messages.

If you experience AWS Cloud Map API throttling, you can contact Support for guidance on how to increase your AWS Cloud Map API throttling limits. For more information about monitoring and troubleshooting such throttling errors, see Handling throttling issues.

Elastic network interfaces

When your tasks use the `awsvpc` network mode, Amazon ECS provisions a unique elastic network interface (ENI) for each task. When your Amazon ECS services use an Elastic Load Balancing load balancer, these network interfaces are also registered as targets to the appropriate target group defined in the service.

Elastic network interface service quotas

When you run tasks that use the `awsvpc` network mode, a unique elastic network interface is attached to each task. If those tasks must be reached over the internet, assign a public IP address to the elastic network interface for those tasks. When you scale your Amazon ECS workloads, consider these two important quotas:

- The `Network interfaces per Region` quota which is the maximum number of network interfaces in an AWS Region for your account.
- The `Elastic IP addresses per Region` quota which is the maximum number of elastic IP addresses in an AWS Region.

Both of these service quotas are adjustable and you can request an increase from your Service Quotas console for these. For more information, see Amazon VPC service quotas.

For Amazon ECS workloads that are hosted on Amazon EC2 instances, when running tasks that use the `awsvpc` network mode consider the `Maximum network interfaces` service quota, the maximum number of network instances for each Amazon EC2 instance. This quota limits the number of tasks that you can place on an instance. You cannot adjust the quota and it's not available in the Service Quotas console. For more information, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide*.

Although you can't change the number of network interfaces that can be attached to an Amazon EC2 instance, you can use the elastic network interface trunking feature to increase the number of available network interfaces. For example, by default a `c5.large` instance can have up to three network interfaces. The primary network interface for the instance counts as one. So, you can attach an additional two network interfaces to the instance. Because each task that uses the `awsvpc` network mode requires a network interface, you can typically only run two such tasks on this instance type. This can lead to under-utilization of your cluster capacity. If you enable elastic network interface trunking, you can increase the network interface density to place a larger number of tasks on each instance. With trunking turned on, a `c5.large` instance can have up to 12 network interfaces. The instance has the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the instance. As a result, with this configuration you can run 10 tasks on the instance instead of the default two tasks. For more information, see Elastic network interface trunking.

Elastic network interface API throttling

When you run tasks that use the `awsvpc` network mode, Amazon ECS relies on the following Amazon EC2 APIs. Each of these APIs have different API throttles. For more information, see Request throttling for the Amazon EC2 API.

- CreateNetworkInterface
- AttachNetworkInterface
- DetachNetworkInterface
- DeleteNetworkInterface
- DescribeNetworkInterfaces
- DescribeVpcs
- DescribeSubnets

- DescribeSecurityGroups
- DescribeInstances

If the Amazon EC2 API calls are throttled during the elastic network interface provisioning workflows, the Amazon ECS service scheduler automatically retries with exponential back-offs. These automatic retries might sometimes lead to a delay in launching tasks, which results in slower deployment speeds. When API throttling occurs, you will see the message `Operations are being throttled. Will try again later.` on your service event messages. If you consistently meet Amazon EC2 API throttles, you can contact Support for guidance on how to increase your API throttling limits. For more information about monitoring and troubleshooting throttling errors, see Handling throttling issues.

AWS Cloud Map

Amazon ECS service discovery uses AWS Cloud Map APIs to manage namespaces for your Amazon ECS services. If your services have a large number of tasks, consider the following recommendations. For more information, see Amazon ECS service discovery considerations.

AWS Cloud Map service quotas

When Amazon ECS services are configured to use service discovery, the `Tasks per service` quota which is the maximum number of tasks for the service, is affected by the AWS Cloud Map `Instances per service` service quota which is the maximum number of instances for that service. In particular, the AWS Cloud Map service quota decreases the amount of tasks that you can run to at most 1,000 tasks per service. You cannot change the AWS Cloud Map quota. For more information, see AWS Cloud Map service quotas.

AWS Cloud Map API throttling

Amazon ECS calls the `ListInstances`, `GetInstancesHealthStatus`, `RegisterInstance`, and `DeregisterInstance` AWS Cloud Map APIs on your behalf. They help with service discovery and perform health checks when you launch a task. When multiple services that use service discovery with a large number of tasks are deployed at the same time, this can result in exceeding the AWS Cloud Map API throttling limits. When this happens, you likely will see the following message: `Operations are being throttled. Will try again later` in your Amazon ECS service event messages and slower deployment and task launch speed. AWS Cloud Map doesn't document throttling limits for these APIs. If you experience throttling from these, you can contact Support for guidance on increasing your API throttling limits. For more recommendations on monitoring and troubleshooting such throttling errors, see Handling throttling issues.

Handle Amazon ECS throttling issues

Throttling errors fall into two major categories: synchronous throttling and asynchronous throttling.

Synchronous throttling

When synchronous throttling occurs, you immediately receive an error response from Amazon ECS. This category typically occurs when you call Amazon ECS APIs while running tasks or creating services. For more information about the throttling involved and the relevant throttle limits, see Request throttling for the Amazon ECS API.

When your application initiates API requests, for example, by using the AWS CLI or an AWS SDK, you can remediate API throttling. You can do this by either architecting your application to handle the errors or by implementing an exponential backoff and jitter strategy with retry logic for the API calls. For more information, see Timeouts, retries, and backoff with jitter.

If you use an AWS SDK, the automatic retry logic is built-in and configurable.

Asynchronous throttling

Asynchronous throttling occurs because of asynchronous workflows where Amazon ECS or AWS CloudFormation might be calling APIs on your behalf to provision resources. It's important to know which AWS APIs that Amazon ECS invokes on your behalf. For example, the `CreateNetworkInterface` API is invoked for tasks that use the `awsvpc` network mode, and the `DescribeTargetHealth` API is invoked when performing health checks for tasks registered to a load balancer.

When your workloads reach a considerable scale, these API operations might be throttled. That is, they might be throttled enough to breach the limits enforced by Amazon ECS or the AWS service that is being called. For example, if you deploy hundreds of services, each having hundreds of tasks concurrently that use the `awsvpc` network mode, Amazon ECS invokes Amazon EC2 API operations such as `CreateNetworkInterface` and Elastic Load Balancing API operations such as `RegisterTarget` or `DescribeTargetHealth` to register the elastic network interface and load balancer, respectively. These API calls can exceed the API limits, resulting in throttling errors. The following is an example of an Elastic Load Balancing throttling error that's included in the service event message.

```
{
   "userIdentity":{
      "arn":"arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForECS/ecs-service-
scheduler",
      "eventTime":"2022-03-21T08:11:24Z",
      "eventSource":"elasticloadbalancing.amazonaws.com",
      "eventName":" DescribeTargetHealth ",
      "awsRegion":"us-east-1",
      "sourceIPAddress":"ecs.amazonaws.com",
      "userAgent":"ecs.amazonaws.com",
      "errorCode":"ThrottlingException",
      "errorMessage":"Rate exceeded",
      "eventID":"0aeb38fc-229b-4912-8b0d-2e8315193e9c"
   }
}
```

When these API calls share limits with other API traffic in your account, they might be difficult monitor even though they're emitted as service events.

Monitor throttling

It's important to identify which API requests are throttled and who issues these requests. You can use AWS CloudTrail which monitors throttling, and integrates with CloudWatch, Amazon Athena, and Amazon EventBridge. You can configure CloudTrail to send specific events to CloudWatch Logs. CloudWatch Logs log insights parses and analyzes the events. This identifies details in throttling events such as the user or IAM role that made the call and the number of API calls that were made. For more information, see [Monitoring CloudTrail log files with CloudWatch Logs](#).

For more information about CloudWatch Logs insights and instructions on how to query log files, see [Analyzing log data with CloudWatch Logs Insights](#).

With Amazon Athena, you can create queries and analyze data using standard SQL. For example, you can create an Athena table to parse CloudTrail events. For more information, see [Using the CloudTrail console to create an Athena table for CloudTrail logs](#).

After creating an Athena table, you can use SQL queries such as the following one to investigate `ThrottlingException` errors.

Replace the *`user-input`* with your values.

```
select eventname, errorcode,eventsource,awsregion, useragent,COUNT(*) count
```

```
FROM cloudtrail_table-name
where errorcode = 'ThrottlingException'
AND eventtime between '2024-09-24T00:00:08Z' and '2024-09-23T23:15:08Z'
group by errorcode, awsregion, eventsource, useragent, eventname
order by count desc;
```

Amazon ECS also emits event notifications to Amazon EventBridge. There are resource state change events and service action events. They include API throttling events such as `ECS_OPERATION_THROTTLED` and `SERVICE_DISCOVERY_OPERATION_THROTTLED`. For more information, see Amazon ECS service action events.

These events can be consumed by a service such as AWS Lambda to perform actions in response. For more information, see Handling Amazon ECS events.

If you run standalone tasks, some API operations such as `RunTask` are asynchronous, and retry operations aren't automatically performed. In such cases, you can use services such as AWS Step Functions with EventBridge integration to retry throttled or failed operations. For more information, see Manage a container task (Amazon ECS, Amazon SNS).

Use CloudWatch to monitor throttling

CloudWatch offers API usage monitoring on the `Usage` namespace under **By AWS Resource**. These metrics are logged with type **API** and metric name **CallCount**. You can create alarms to start whenever these metrics reach a certain threshold. For more information, see Visualizing your service quotas and setting alarms.

CloudWatch also offers anomaly detection. This feature uses machine learning to analyze and establish baselines based on the particular behavior of the metric that you enabled it on. If there's unusual API activity, you can use this feature together with CloudWatch alarms. For more information, see Using CloudWatch anomaly detection.

By proactively monitoring throttling errors, you can contact Support to increase the relevant throttling limits and also receive guidance for your unique application needs.

Amazon ECS Auto scaling and capacity management best practices

You can run containerized application workloads of all sizes on Amazon ECS. This includes minimal testing environments and large production environments that operate at a global scale.

With Amazon ECS, like all AWS services, you pay only for what you use. When you architect your application appropriately, you can save costs by consuming only the resources that you need when you need them.

The following recommendations show you how to run your Amazon ECS workloads to meet your service-level objectives while operating cost-effectively.

Topics

Determining the task size for Amazon ECS

One of the most important choices when you deploy containers on Amazon ECS is your container and task sizes. Your container and task sizes are essential for scaling and capacity planning.

Amazon ECS uses two resource metrics for capacity: CPU and memory. Amazon ECS measures CPU in units of 1/1024 of a full vCPU (where 1024 units equals 1 whole vCPU). Amazon ECS measures memory in megabytes.

In your task definition, you can declare resource reservations and limits.

When you declare a reservation, you declare the minimum amount of resources that a task requires. Your task receives at least the amount of resources you request. Your application might be able to use more CPU or memory than the reservation you declare. However, this is subject to any limits you also declared.

Using more than the reservation amount is known as bursting. Bursting means your application uses more resources than you reserved but stays within your declared limits. Amazon ECS guarantees reservations. For example, if you use Amazon EC2 instances to provide capacity, Amazon ECS doesn't place a task on an instance where it can't fulfill the reservation.

A limit is the maximum amount of CPU units or memory that your container or task can use. If your container tries to use more CPU than this limit, Amazon ECS throttles it. If your container tries to use more memory than this limit, Amazon ECS stops your container.

Choosing these values can be challenging. The values that work best for your application depend greatly on your application's resource requirements.

Load testing your application is the key to successful resource requirement planning. Load testing helps you better understand your application's requirements.

Stateless applications

For stateless applications that scale horizontally, such as an application behind a load balancer, we recommend that you first determine how much memory your application consumes when it serves requests.

To do this, you can use traditional tools such as `ps` or `top`. You can also use monitoring solutions such as CloudWatch Container Insights.

When you determine a CPU reservation, consider how you want to scale your application to meet your business requirements.

You can use smaller CPU reservations, such as 256 CPU units (or 1/4 vCPU), to scale out in a fine-grained way that minimizes cost. But they might not scale fast enough to meet significant spikes in demand.

You can use larger CPU reservations to scale in and out more quickly. This helps you match demand spikes more quickly. However, larger CPU reservations cost more.

Other applications

For applications that don't scale horizontally, such as singleton workers or database servers, available capacity and cost are your most important considerations.

Choose the amount of memory and CPU based on what load testing shows you need to serve traffic and meet your service-level objective. Amazon ECS ensures that your application is placed on a host that has adequate capacity.

Optimizing Amazon ECS service auto scaling

An Amazon ECS service is a managed collection of tasks. Each service has an associated task definition, a desired task count, and an optional placement strategy.

Amazon ECS service auto scaling works through the Application Auto Scaling service. Application Auto Scaling uses CloudWatch metrics as the source for scaling metrics. It also uses CloudWatch alarms to set thresholds on when to scale your service in or out.

You provide the thresholds for scaling. You can set a metric target, which is called *target tracking scaling*. You can also specify thresholds, which is called *step scaling*.

After you configure Application Auto Scaling, it continually calculates the appropriate desired task count for the service. It also notifies Amazon ECS when the desired task count should change, either by scaling it out or scaling it in.

To use service auto scaling effectively, you must choose an appropriate scaling metric. We discuss how to choose a metric in the following sections.

Characterizing your application

Properly scaling an application requires you to know the conditions where you should scale your application in and when you should scale it out.

In essence, you should scale out your application if demand is forecasted to outstrip capacity. Conversely, you can scale in your application to conserve costs when resources exceed demand.

Identifying a utilization metric

To scale effectively, you must identify a metric that indicates utilization or saturation. This metric must exhibit the following properties to be useful for scaling.

- The metric must be correlated with demand. When you hold resources steady but demand changes, the metric value must also change. The metric should increase or decrease when demand increases or decreases.
- The metric value must scale in proportion to capacity. When demand holds constant, adding more resources must result in a proportional change in the metric value. So, doubling the number of tasks should cause the metric to decrease by 50%.

The best way to identify a utilization metric is through load testing in a pre-production environment such as a staging environment. Commercial and open-source load testing solutions are widely available. These solutions typically can either generate synthetic load or simulate real user traffic.

To start the process of load testing, you should first build dashboards for your application’s utilization metrics. These metrics include CPU utilization, memory utilization, I/O operations, I/O queue depth, and network throughput. You can collect these metrics with a service such as CloudWatch Container Insights. You can also collect them by using Amazon Managed Service for Prometheus together with Amazon Managed Grafana. During this process, make sure that you collect and plot metrics for your application’s response times or work completion rates.

When you load test, begin with a small request or job insertion rate. Keep this rate steady for several minutes to allow your application to warm up. Then, slowly increase the rate and hold it steady for a few minutes. Repeat this cycle, increasing the rate each time until your application’s response or completion times are too slow to meet your service-level objectives (SLOs).

While you load test, examine each of the utilization metrics. The metrics that increase along with the load are the top candidates to serve as your best utilization metrics.

Next, identify the resource that reaches saturation. At the same time, also examine the utilization metrics to see which one flattens out at a high level first. Or, examine which one reaches peak and then crashes your application first. For example, if CPU utilization increases from 0% to 70-80% as you add load, then stays at that level after you add even more load, then it's safe to say that the CPU is saturated. Depending on the CPU architecture, it might never reach 100%. For example, assume that memory utilization increases as you add load, and then your application suddenly crashes when it reaches the task or Amazon EC2 instance memory limit. In this situation, it's likely the case that memory has been fully consumed. Multiple resources might be consumed by your application. Therefore, choose the metric that represents the resource that depletes first.

Last, try load testing again after you double the number of tasks or Amazon EC2 instances. Assume that the key metric increases, or decreases, at half the rate as before. If this is the case, then the metric is proportional to capacity. This is a good utilization metric for auto scaling.

Now consider this hypothetical scenario. Suppose that you load test an application and find that the CPU utilization eventually reaches 80% at 100 requests per second. When you add more load, it doesn't make CPU utilization increase anymore. However, it does make your application respond more slowly. Then, you run the load test again, doubling the number of tasks but holding the rate at its previous peak value. If you find the average CPU utilization falls to about 40%, then average CPU utilization is a good candidate for a scaling metric. On the other hand, if CPU utilization remains at 80% after increasing the number of tasks, then average CPU utilization isn't a good scaling metric. In that case, you need more research to find a suitable metric.

Common application models and scaling properties

You can run software of all kinds on AWS. Many workloads are homegrown, whereas others are based on popular open-source software. Regardless of where they originate, we have observed some common design patterns for services. How you scale effectively depends in large part on the pattern.

The efficient CPU-bound server

The efficient CPU-bound server utilizes almost no resources other than CPU and network throughput. Each request can be handled by the application alone. Requests don't depend on other services such as databases. The application can handle hundreds of thousands of concurrent requests, and can efficiently utilize multiple CPUs to do so. Each request is either serviced by a dedicated thread with low memory overhead, or there's an asynchronous event loop that runs on each CPU that services requests. Each replica of the application is equally capable of handling a request. The only resource that might be depleted before CPU is network bandwidth. In CPU bound-services, memory utilization, even at peak throughput, is a fraction of the resources available.

You can use CPU-based auto scaling for this type of application. The application enjoys maximum flexibility in terms of scaling. You can scale it vertically by providing larger Amazon EC2 instances or Fargate vCPUs to it. And, you can also scale it horizontally by adding more replicas. Adding more replicas, or doubling the instance size, cuts the average CPU utilization relative to capacity by half.

If you're using Amazon EC2 capacity for this application, consider placing it on compute-optimized instances such as the `c5` or `c6g` family.

The efficient memory-bound server

The efficient memory-bound server allocates a significant amount of memory per request. At maximum concurrency, but not necessarily throughput, memory is depleted before the CPU resources are depleted. Memory associated with a request is freed when the request ends. Additional requests can be accepted as long as there is available memory.

You can use memory-based auto scaling for this type of application. The application enjoys maximum flexibility in terms of scaling. You can scale it both vertically by providing larger Amazon EC2 or Fargate memory resources to it. And, you can also scale it horizontally by adding more replicas. Adding more replicas, or doubling the instance size, can cut the average memory utilization relative to capacity by half.

If you're using Amazon EC2 capacity for this application, consider placing it on memory-optimized instances such as the `r5` or `r6g` family.

Some memory-bound applications don't free the memory that's associated with a request when it ends, so that a reduction in concurrency doesn't result in a reduction in the memory used. For this, we don't recommend that you use memory-based scaling.

The worker-based server

The worker-based server processes one request for each individual worker thread one after another. The worker threads can be lightweight threads, such as POSIX threads. They can also be heavier-weight threads, such as UNIX processes. No matter which thread they are, there's always a maximum concurrency that the application can support. Usually the concurrency limit is set proportionally to the memory resources that are available. If the concurrency limit is reached, the application places additional requests into a backlog queue. If the backlog queue overflows, the application immediately rejects additional incoming requests. Common applications that fit this pattern include Apache web server and Gunicorn.

Request concurrency is usually the best metric for scaling this application. Because there's a concurrency limit for each replica, it's important to scale out before the average limit is reached.

The best way to obtain request concurrency metrics is to have your application report them to CloudWatch. Each replica of your application can publish the number of concurrent requests as a custom metric at a high frequency. We recommend that the frequency is set to be at least once every minute. After several reports are collected, you can use the average concurrency as a scaling metric. You calculate this metric by taking the total concurrency and dividing it by the number of replicas. For example, if total concurrency is 1000 and the number of replicas is 10, then the average concurrency is 100.

If your application is behind an Application Load Balancer, you can also use the `ActiveConnectionCount` metric for the load balancer as a factor in the scaling metric. You must divide the `ActiveConnectionCount` metric by the number of replicas to obtain an average value. You must use the average value for scaling, as opposed to the raw count value.

For this design to work best, the standard deviation of response latency should be small at low request rates. We recommend that, during periods of low demand, most requests are answered within a short time, and there isn't a lot of requests that take significantly longer than average time to respond. The average response time should be close to the 95th percentile response time. Otherwise, queue overflows might occur as result. This leads to errors. We recommend that you provide additional replicas where necessary to mitigate the risk of overflow.

The waiting server

The waiting server does some processing for each request, but it is highly dependent on one or more downstream services to function. Container applications often make heavy use of downstream services like databases and other API services. It can take some time for these services to respond, particularly in high capacity or high concurrency scenarios. This is because these applications tend to use few CPU resources and utilize their maximum concurrency in terms of available memory.

The waiting service is suitable either in the memory-bound server pattern or the worker-based server pattern, depending on how the application is designed. If the application’s concurrency is limited only by memory, then average memory utilization should be used as a scaling metric. If the application’s concurrency is based on a worker limit, then average concurrency should be used as a scaling metric.

The Java-based server

If your Java-based server is CPU-bound and scales proportionally to CPU resources, then it might be suitable for the efficient CPU-bound server pattern. If that is the case, average CPU utilization might be appropriate as a scaling metric. However, many Java applications aren't CPU-bound, making them challenging to scale.

For the best performance, we recommend that you allocate as much memory as possible to the Java Virtual Machine (JVM) heap. Recent versions of the JVM, including Java 8 update 191 or later, automatically set the heap size as large as possible to fit within the container. This means that, in Java, memory utilization is rarely proportional to application utilization. As the request rate and concurrency increases, memory utilization remains constant. Because of this, we don't recommend scaling Java-based servers based on memory utilization. Instead, we typically recommend scaling on CPU utilization.

In some cases, Java-based servers encounter heap exhaustion before exhausting CPU. If your application is prone to heap exhaustion at high concurrency, then average connections are the best scaling metric. If your application is prone to heap exhaustion at high throughput, then average request rate is the best scaling metric.

Servers that use other garbage-collected runtimes

Many server applications are based on runtimes that perform garbage collection such as .NET and Ruby. These server applications might fit into one of the patterns described earlier. However, as with Java, we don't recommend scaling these applications based on memory, because their observed average memory utilization is often uncorrelated with throughput or concurrency.

For these applications, we recommend that you scale on CPU utilization if the application is CPU bound. Otherwise, we recommend that you scale on average throughput or average concurrency, based on your load testing results.

Job processors

Many workloads involve asynchronous job processing. They include applications that don't receive requests in real time, but instead subscribe to a work queue to receive jobs. For these types of applications, the proper scaling metric is almost always queue depth. Queue growth is an indication that pending work outstrips processing capacity, whereas an empty queue indicates that there's more capacity than work to do.

AWS messaging services, such as Amazon SQS and Amazon Kinesis Data Streams, provide CloudWatch metrics that can be used for scaling. For Amazon SQS, `ApproximateNumberOfMessagesVisible` is the best metric. For Kinesis Data Streams, consider using the `MillisBehindLatest` metric, published by the Kinesis Client Library (KCL). This metric should be averaged across all consumers before using it for scaling.

Amazon ECS capacity and availability

Application availability is crucial for providing an error-free experience and for minimizing application latency. Availability depends on having resources that are accessible and have enough capacity to meet demand. AWS provides several mechanisms to manage availability. For applications that you host on Amazon ECS, these include autoscaling and Availability Zones (AZs). Autoscaling manages the number of tasks or instances based on metrics you define, while Availability Zones allow you to host your application in isolated but geographically-close locations.

As with task sizes, capacity and availability present certain trade-offs you must consider. Ideally, capacity would be perfectly aligned with demand. There would always be just enough capacity to serve requests and process jobs to meet Service Level Objectives (SLOs) including a low latency and error rate. Capacity would never be too high, leading to excessive cost; nor would it never be too low, leading to high latency and error rates.

Autoscaling is a latent process. First, CloudWatch must receive real-time metrics. Then, CloudWatch needs to aggregate them for analysis, which can take up to several minutes depending on the granularity of the metric. CloudWatch compares the metrics against alarm thresholds to identify a shortage or excess of resources. To prevent instability, you should configure alarms to require the set threshold be crossed for a few minutes before the alarm goes off. It also takes time to provision new tasks and to terminate tasks that are no longer needed.

Because of these potential delays in the system, you should maintain some headroom by over-provisioning. Over-provisioning can help accommodate short-term bursts in demand. This also helps your application service additional requests without reaching saturation. As a good practice, you can set your scaling target between 60-80% of utilization. This helps your application better handle bursts of extra demand while additional capacity is still in the process of being provisioned.

Another reason we recommend that you over-provision is so that you can quickly respond to Availability Zone failures. AWS recommends that production workloads be served from multiple Availability Zones. This is because, if an Availability Zone failure occurs, your tasks that are running in the remaining Availability Zones can still serve the demand. If your application runs in two Availability Zones, you need to double your normal task count. This is so that you can provide immediate capacity during any potential failure. If your application runs in three Availability Zones, we recommend that you run 1.5 times your normal task count. That is, run three tasks for every two that are needed for ordinary serving.

Maximizing scaling speed

Autoscaling is a reactive process that takes time to take effect. However, there are some ways to help minimize the time that's needed to scale out.

Minimize image size. Larger images take longer to download from an image repository and unpack. Therefore, keeping image sizes smaller reduces the amount of time that's needed for a container to start. To reduce the image size, you can follow these specific recommendations:

- If you can build a static binary or use Golang, build your image `FROM` scratch and include only your binary application in the resulting image.
- Use minimized base images from upstream distro vendors, such as Amazon Linux or Ubuntu.
- Don’t include any build artifacts in your final image. Using multi-stage builds can help with this.
- Compact `RUN` stages wherever possible. Each `RUN` stage creates a new image layer, leading to an additional round trip to download the layer. A single `RUN` stage that has multiple commands joined by `&&` has fewer layers than one with multiple `RUN` stages.
- If you want to include data, such as ML inference data, in your final image, include only the data that's needed to start up and begin serving traffic. If you fetch data on demand from Amazon S3 or other storage without impacting service, then store your data in those places instead.

Keep your images close. The higher the network latency, the longer it takes to download the image. Host your images in a repository in the same AWS Region that your workload is in. Amazon ECR is a high-performance image repository that's available in every Region that Amazon ECS is available in. Avoid traversing the Internet or a VPN link to download container images. Hosting your images in the same Region improves overall reliability. It mitigating the risk of network connectivity issues and availability issues in a different Region. Alternatively, you can also implement Amazon ECR cross-region replication to help with this.

Reduce load balancer health check thresholds. Load balancers perform health checks before sending traffic to your application. The default health check configuration for a target group can take 90 seconds or longer. During this, the load balancer checks the health status and receives requests. Lowering the health check interval and threshold count can make your application accept traffic quicker and reduce load on other tasks.

Consider cold-start performance. Some application use runtimes such as Java perform Just-In-Time (JIT) compilation. The compilation process at least as it starts can show application performance. A workaround is to rewrite the latency-critical parts of your workload in languages that don't impose a cold-start performance penalty.

Use step scaling, not target-tracking scaling policies. You have several Application Auto Scaling options for Amazon ECS tasks. Target tracking is the easiest mode to use. With it, all you need to do is set a target value for a metric, such as CPU average utilization. Then, the auto scaler automatically manages the number of tasks that are needed to attain that value. With step scaling you can more quickly react to changes in demand, because you define the specific thresholds for your scaling metrics, and how many tasks to add or remove when the thresholds are crossed. And, more importantly, you can react very quickly to changes in demand by minimizing the amount of time a threshold alarm is in breach. For more information, see Service Auto Scaling in the *Amazon Elastic Container Service Developer Guide*.

If you're using Amazon EC2 instances to provide cluster capacity, consider the following recommendations:

Use larger Amazon EC2 instances and faster Amazon EBS volumes. You can improve image download and preparation speeds by using a larger Amazon EC2 instance and faster Amazon EBS volume. Within a given Amazon EC2 instance family, the network and Amazon EBS maximum throughput increases as the instance size increases (for example, from `m5.xlarge` to `m5.2xlarge`). Additionally, you can also customize Amazon EBS volumes to increase their throughput and IOPS. For example, if you're using gp2 volumes, use larger volumes that offer more baseline throughput. If you're using gp3 volumes, specify throughput and IOPS when you create the volume.

Use bridge network mode for tasks running on Amazon EC2 instances. Tasks that use bridge network mode on Amazon EC2 start faster than tasks that use the awsvpc network mode. When awsvpc network mode is used, Amazon ECS attaches an elastic network interface (ENI) to the instance before launching the task. This introduces additional latency. There are several tradeoffs for using bridge networking though. These tasks don't get their own security group, and there are some implications for load balancing. For more information, see Load balancer target groups in the *Elastic Load Balancing User Guide*.

Handling demand shocks

Some applications experience sudden large shocks in demand. This happens for a variety of reasons: a news event, big sale, media event, or some other event that goes viral and causes traffic to quickly and significantly increase in a very short span of time. If unplanned, this can cause demand to quickly outstrip available resources.

The best way to handle demand shocks is to anticipate them and plan accordingly. Because autoscaling can take time, we recommend that you scale out your application before the demand shock begins. For the best results, we recommend having a business plan that involves tight collaboration between teams that use a shared calender. The team that's planning the event should work closely with the team in charge of the application in advance. This gives that team enough time to have a clear scheduling plan. They can schedule capacity to scale out before the event and to scale in after the event. For more information, see Scheduled scaling in the *Application Auto Scaling User Guide*.

If you have an Enterprise Support plan, be sure also to work with your Technical Account Manager (TAM). Your TAM can verify your service quotas and ensure that any necessary quotas are raised before the event begins. This way, you don't accidentally hit any service quotas. They can also help you by prewarming services such as load balancers to make sure your event goes smoothly.

Handling unscheduled demand shocks is a more difficult problem. Unscheduled shocks, if large enough in amplitude, can quickly cause demand to outstrip capacity. It can also outpace the ability for autoscaling to react. The best way to prepare for unscheduled shocks is to over-provision resources. You must have enough resources to handle maximum anticipated traffic demand at any time.

Maintaining maximum capacity in anticipation of unscheduled demand shocks can be costly. To mitigate the cost impact, find a leading indicator metric or event that predicts a large demand shock is imminent. If the metric or event reliably provides significant advance notice, begin the scale-out process immediately when the event occurs or when the metric crosses the specific threshold that you set.

If your application is prone to sudden unscheduled demand shocks, consider adding a high-performance mode to your application that sacrifices non-critical functionality but retains crucial functionality for a customer. For example, assume that your application can switch from generating expensive customized responses to serving a static response page. In this scenario, you can increase throughput significantly without scaling the application at all.

Last, you can consider breaking apart monolithic services to better deal with demand shocks. If your application is a monolithic service that's expensive to run and slow to scale, you might be able to extract or rewrite performance-critical pieces and run them as separate services. These new services then can be scaled independently from less-critical components. Having the flexibility to scale out performance-critical functionality separately from other parts of your application can both reduce the time it takes to add capacity and help conserve costs.

Amazon ECS cluster capacity

You can provide capacity to an Amazon ECS cluster in several ways. For example, you can launch Amazon EC2 instances and register them with the cluster at start-up using the Amazon ECS container agent. However, this method can be challenging because you need to manage scaling on your own. Therefore, we recommend that you use Amazon ECS capacity providers. Capacity providers manage resource scaling for you. There are three kinds of capacity providers: Amazon EC2, Fargate, and Fargate Spot. For more information about Fargate capacity providers, see Amazon ECS clusters for Fargate launch type workloads and for the EC2 launch type see Amazon ECS clusters for EC2 launch type workloads.

The Fargate and Fargate Spot capacity providers handle the lifecycle of Fargate tasks for you. Fargate provides on-demand capacity, and Fargate Spot provides Spot capacity. When you launch a task, Amazon ECS provisions a Fargate resource for you. This Fargate resource comes with the memory and CPU units that directly correspond to the task-level limits that you declared in your task definition. Each task receives its own Fargate resource, making a 1:1 relationship between the task and compute resources.

Tasks that run on Fargate Spot are subject to interruption. Interruptions come after a two-minute warning. These occur during periods of heavy demand. Fargate Spot works best for interruption-tolerant workloads such as batch jobs, development or staging environments. They're also suitable for any other scenario where high availability and low latency isn't a requirement.

You can run Fargate Spot tasks alongside Fargate on-demand tasks. By using them together, you receive provision "burst" capacity at a lower cost.

Amazon ECS can also manage the Amazon EC2 instance capacity for your tasks. Each Amazon EC2 capacity provider is associated with an Amazon EC2 Auto Scaling group that you specify. When you use the Amazon EC2 capacity provider, cluster auto scaling maintains the size of the Amazon EC2 Auto Scaling group to ensure all scheduled tasks can be placed.

Choosing Fargate task sizes for Amazon ECS

For AWS Fargate task definitions, you're required to specify CPU and memory at the task level, and do not need to account for any overhead. You can also specify CPU and memory at the container level for Fargate tasks. However, doing so isn't required. The resource limits must be greater than or equal to any reservations that you declared. In most cases, you can set them to the sum of the reservations of each of the container that's declared in your task definition. Then, round the number up to the nearest Fargate size. For more information about the available sizes, see Task CPU and memory.

Speeding up Amazon ECS cluster capacity provisioning with capacity providers on Amazon EC2

Customers who run Amazon ECS on Amazon EC2 can take advantage of Amazon ECS Cluster Auto Scaling (CAS) to manage the scaling of Amazon EC2 Auto Scaling groups (ASG). With CAS, you can configure Amazon ECS to scale your ASG automatically, and just focus on running your tasks. Amazon ECS will ensure the ASG scales in and out as needed with no further intervention required. Amazon ECS capacity providers are used to manage the infrastructure in your cluster by ensuring there are enough container instances to meet the demands of your application. To learn how Amazon ECS CAS works, see Deep Dive on Amazon ECS Cluster Auto Scaling.

Since CAS relies on a CloudWatch based integration with ASG for adjusting cluster capacity, it has inherent latency associated with publishing the CloudWatch metrics, the time taken for the metric `CapacityProviderReservation` to breach CloudWatch alarms (both high and low), and the time taken by a newly launched Amazon EC2 instance to warm-up. You can take the following actions to make CAS more responsive for faster deployments:

Capacity provider step scaling sizes

Amazon ECS capacity providers will eventually grow/shrink the container instances to meet the demands of your application. The minimum number of instances that Amazon ECS will launch is set to 1 by default. This may add additional time to your deployments, if several instances are required for placing your pending tasks. You can increase the `minimumScalingStepSize` by using the Amazon ECS API to increase the minimum number of instances that Amazon ECS scales in or out at a time. A `maximumScalingStepSize` that is too low can limit how many container instances are scaled in or out at a time, which can slow down your deployments.

> **Note**
>
> This configuration is currently only available by using the `CreateCapacityProvider` or `UpdateCapacityProvider` APIs.

Instance warm-up period

The instance warm-up period is the period of time after which a newly launched Amazon EC2 instance can contribute to CloudWatch metrics for the Auto Scaling group. After the specified warm-up period expires, the instance is counted toward the aggregated metrics of the ASG, and CAS proceeds with its next iteration of calculations to estimate the number instances required.

The default value for `instanceWarmupPeriod` is 300 seconds, which you can configure to a lower value by using the `CreateCapacityProvider` or `UpdateCapacityProvider` APIs for more responsive scaling.

Spare capacity

If your capacity provider has no container instances available for placing tasks, then it needs to increase (scale out) cluster capacity by launching Amazon EC2 instances on the fly, and wait for them to boot up before it can launch containers on them. This can significantly lower the task launch rate. You have two options here.

In this case, having spare Amazon EC2 capacity already launched and ready to run tasks will increase the effective task launch rate. You can use the `Target Capacity` configuration to indicate that you wish to maintain spare capacity in your clusters. For example, by setting `Target Capacity` at 80%, you indicate that your cluster needs 20% spare capacity at all times. This spare capacity can allow any standalone tasks to be immediately launched, ensuring task launches are not throttled. The trade-off for this approach is potential increased costs of keeping spare cluster capacity.

An alternate approach you can consider is adding headroom to your service, not to the capacity provider. This means that instead of reducing `Target Capacity` configuration to launch spare capacity, you can increase the number of replicas in your service by modifying the target tracking scaling metric or the step scaling thresholds of the service auto scaling. Note that this approach will only be helpful for spiky workloads, but won't have an effect when you're deploying new services and going from 0 to N tasks for the first time. For more information about the related scaling policies, see Target Tracking Scaling Policies or Step Scaling Policies

Access Amazon ECS features with account settings

You can go into Amazon ECS account settings to opt in or out of specific features. For each AWS Region, you can opt in to, or opt out of, each account setting at the account-level or for a specific user or role.

You might want to opt in or out of specific features if any of the following is relevant to you:

- A user or role can opt in or opt out specific account settings for their individual account.
- A user or role can set the default opt-in or opt-out setting for all users on the account.
- The root user or a user with administrator privileges can opt in to, or opt out of, any specific role or user on the account. If the account setting for the root user is changed, it sets the default for all the users and roles that no individual account setting was selected for.

Note

Federated users assume the account setting of the root user and can't have explicit account settings set for them separately.

The following account settings are available. You must separately opt-in and opt-out to each account setting.

Resource name	Learn more
`containerInsights`	Container Insights
`serviceLongArnFormat` `taskLongArnFormat`	Amazon Resource Names (ARNs) and IDs

Resource name	Learn more
containerInstanceLongArnFormat	
tagResourceAuthorization	Tagging authorization
fargateFIPSMode	AWS Fargate Federal Information Processing Standard (FIPS-140) compliance
fargateTaskRetirementWaitPeriod	AWS Fargate task retirement wait time
guardDutyActivate	Runtime Monitoring (Amazon GuardDuty integration)
dualStackIPv6	Dual stack IPv6 VPC
awsvpcTrunking	Increase Linux container instance network interfaces
defaultLogDriverMode	Default log driver mode

Amazon Resource Names (ARNs) and IDs

When Amazon ECS resources are created, each resource is assigned a unique Amazon Resource Name (ARN) and resource identifier (ID). If you use a command line tool or the Amazon ECS API to work with Amazon ECS, resource ARNs or IDs are required for certain commands. For example, if you use the stop-task AWS CLI command to stop a task, you must specify the task ARN or ID in the command.

Amazon ECS introduced a new format for Amazon Resource Names (ARNs) and resource IDs for Amazon ECS services, tasks, and container instances. The opt-in status for each resource type determines the Amazon Resource Name (ARN) format the resource uses. You must opt in to the new ARN format to use features such as resource tagging for that resource type.

You can opt in to and opt out of the new Amazon Resource Name (ARN) and resource ID format on a per-Region basis. Currently, any new account created is opted in by default.

You can opt in or opt out of the new Amazon Resource Name (ARN) and resource ID format at any time. After you opt in, any new resources that you create use the new format.

> **Note**
>
> A resource ID doesn't change after it's created. Therefore, opting in or out of the new format doesn't affect your existing resource IDs.

The following sections describe how ARN and resource ID formats are changing. For more information about the transition to the new formats, see [Amazon Elastic Container Service FAQ](#).

Amazon Resource Name (ARN) format

Some resources have a user-friendly name, such as a service named `production`. In other cases, you must specify a resource using the Amazon Resource Name (ARN) format. The new ARN format for Amazon ECS tasks, services, and container instances includes the cluster name. For information about opting in to the new ARN format, see [Modifying Amazon ECS account settings](#).

The following table shows both the current format and the new format for each resource type.

Resource type	ARN
Container instance	Current: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:container-instance/` *`container-instance-id`* New: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:container-instance/` *`cluster-name`* `/`*`container-instance-id`*
Amazon ECS service	Current: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:service/` *`service-name`* New: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:service/` *`cluster-name`* `/`*`service-name`*
Amazon ECS task	Current: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:task/`*`task-id`* New: `arn:aws:ecs:` *`region`*`:`*`aws_account_id`* `:task/`*`cluster-name`* `/`*`task-id`*

Resource ID length

A resource ID takes the form of a unique combination of letters and numbers. New resource ID formats include shorter IDs for Amazon ECS tasks and container instances. The current resource ID format is 36 characters long. The new IDs are in a 32-character format that doesn't include any hyphens. For information about opting in to the new resource ID format, see [Modifying Amazon ECS account settings](#).

The default is `enabled`.

Only resources launched after opting in receive the new ARN and resource ID format. All existing resources aren't affected. For Amazon ECS services and tasks to transition to the new ARN and resource ID formats, you must recreate the service or task. To transition a container instance to the new ARN and resource ID format, the container instance must be drained and a new container instance must be launched and registered to the cluster.

> **Note**
>
> Tasks launched by an Amazon ECS service can only receive the new ARN and resource ID format if the service was created on or after November 16, 2018, and the user who created the service has opted in to the new format for tasks.

ARN and resource ID format timeline

The timeline for the opt-in and opt-out periods for the new Amazon Resource Name (ARN) and resource ID format for Amazon ECS resources ended on April 1, 2021. By default, all accounts are opted in to the new format. All new resources created receive the new format, and you can no longer opt out.

Container Insights

On December 2, 2024, AWS released Container Insights with enhanced observability for Amazon ECS. This version supports enhanced observability for Amazon ECS clusters using the Amazon EC2 and Fargate launch types. After you configure Container Insights with enhanced observability on Amazon ECS, Container Insights auto-collects detailed infrastructure telemetry from cluster level down to the container level in your environment and displays your data in dashboards that show you a variety of metrics and dimensions. You can then use these out-of-the-box dashboards on the Container Insights console to better understand your container health and performance, and to mitigate issues faster by identifying anomalies.

We recommend that you use Container Insights with enhanced observability instead of Container Insights because it provides detailed visibility in your container environment, reducing the mean time to resolution. For more information, see [Amazon ECS Container Insights with enhanced observability metrics]() in the *Amazon CloudWatch User Guide*.

The default for the `containerInsights` account setting is `disabled`.

Container Insights with enhanced observability

Use the following command to turn on Container Insights with enhanced observability.

Set the `containerInsights` account setting to `enhanced`.

```
aws ecs put-account-setting --name containerInsights --value enhanced
```

Example output

```
{
    "setting": {
        "name": "containerInsights",
        "value": "enhanced",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
    }
}
```

After you set this account setting, all new clusters automatically use Container Insights with enhanced observability. Use the `update-cluster-settings` command to add Container Insights with enhanced observability to an existing cluster, or to upgrade a cluster from Container Insights to Container Insights with enhanced observability.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enhanced
```

You can also use the console to configure Container Insights with enhanced observability. For more information, see [Modifying Amazon ECS account settings]().

Container Insights

When you set the `containerInsights` account setting to `enabled`, all new clusters have Container Insights enabled by default. You can modify existing clusters by using `update-cluster-settings`.

To use Container Insights, set the `containerInsights` account setting to `enabled`. Use the following command to turn on Container Insights.

```
aws ecs put-account-setting --name containerInsights --value enabled
```

Example output

```
{
    "setting": {
        "name": "containerInsights",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:johndoe",
         "type": user
    }
}
```

When you set the `containerInsights` account setting to `enabled`, all new clusters have Container Insights enabled by default. Use the `update-cluster-settings` command to add Container Insights to an existing cluster.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enabled
```

You can also use the console to configure Container Insights. For more information, see Modifying Amazon ECS account settings.

AWS Fargate Federal Information Processing Standard (FIPS-140) compliance

Fargate supports the Federal Information Processing Standard (FIPS-140) which specifies the security requirements for cryptographic modules that protect sensitive information. It is the current United States and Canadian government standard, and is applicable to systems that are required to be compliant with Federal Information Security Management Act (FISMA) or Federal Risk and Authorization Management Program (FedRAMP).

The resource name is `fargateFIPSMode`.

The default is `disabled`.

You must turn on Federal Information Processing Standard (FIPS-140) compliance on Fargate. For more information, see the section called “AWS Fargate FIPS-140 compliance”.

Important

The `fargateFIPSMode` account setting can only be changed using either the Amazon ECS API or the AWS CLI. For more information, see Modifying Amazon ECS account settings.

Run `put-account-setting-default` with the `fargateFIPSMode` option set to `enabled`. For more information, see, put-account-setting-default in the *Amazon Elastic Container Service API Reference*.

- You can use the following command to turn on FIPS-140 compliance.

```
aws ecs put-account-setting-default --name fargateFIPSMode --value enabled
```

Example output

```
{
    "setting": {
        "name": "fargateFIPSMode",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:root",
         "type": user
    }
}
```

You can run `list-account-settings` to view the current FIPS-140 compliance status. Use the `effective-settings` option to view the account level settings.

```
aws ecs list-account-settings --effective-settings
```

Tagging authorization

Amazon ECS is introducing tagging authorization for resource creation. Users must have tagging permissions for actions that create the resource, such as `ecsCreateCluster`. When you create a resource and specify tags for that resource, AWS performs additional authorization to verify that there are permissions to create tags. Therefore, you must grant explicit permissions to use the `ecs:TagResource` action. For more information, see the section called "Tag resources during creation".

In order to opt in to tagging authorization, run `put-account-setting-default` with the `tagResourceAuthorization` option set to `enable`. For more information, see, put-account-setting-default in the *Amazon Elastic Container Service API Reference*. You can run `list-account-settings` to view the current tagging authorization status.

- You can use the following command to enable tagging authorization.

  ```
  aws ecs put-account-setting-default --name tagResourceAuthorization --value on --
  region region
  ```

  Example output

  ```
  {
      "setting": {
          "name": "tagResourceAuthorization",
          "value": "on",
          "principalArn": "arn:aws:iam::123456789012:root",
          "type": user
      }
  }
  ```

After you enable tagging authorization, you must configure the appropriate permissions to allow users to tag resources on creation. For more information, see the section called "Tag resources during creation".

You can run `list-account-settings` to view the current tagging authorization status. Use the `effective-settings` option to view the account level settings.

```
aws ecs list-account-settings --effective-settings
```

Tagging authorization timeline

You can confirm whether tagging authorization is active by running `list-account-settings` to view the `tagResourceAuthorization` value. When the value is `on`, it means that the tagging authorization is in use. For more information, see, list-account-settings in the *Amazon Elastic Container Service API Reference*.

The following are the important dates related to tagging authorization.

- April 18, 2023 – Tagging authorization is introduced. All new and existing accounts must opt in to use the feature. You can opt in to start using tagging authorization. By opting in, you must grant the appropriate permissions.
- February 9, 2024 - March 6, 2024 – All new accounts and non-impacted existing accounts have tagging authorization on by default. You can enable or disable the `tagResourceAuthorization` account setting to verify your IAM policy.

  AWS has notified impacted accounts.

  To disable the feature, run `put-account-setting-default` with the `tagResourceAuthorization` option set to `off`.
- March 7, 2024 – If you have enabled tagging authorization, you can no longer disable the account setting.

  We recommend that you complete your IAM policy testing before this date.
- March 29, 2024 – All accounts use tagging authorization. The account-level setting will no longer be available in the Amazon ECS console or AWS CLI.

AWS Fargate task retirement wait time

AWS sends out notifications when you have Fargate tasks running on a platform version revision marked for retirement. For more information, see Task retirement and maintenance for AWS Fargate on Amazon ECS .

AWS is responsible for patching and maintaining the underlying infrastructure for AWS Fargate. When AWS determines that a security or infrastructure update is needed for an Amazon ECS task hosted on Fargate, the tasks need to be stopped and new tasks launched to replace them. You can configure the wait period before tasks are retired for patching. You have the option to retire the task immediately, to wait 7 calendar days, or to wait 14 calendar days.

This setting is at the account-level.

You can configure the time that Fargate starts the task retirement. For workloads that require immediate application of the updates, choose the immediate setting (`0`). When you need more control, for example, when a task can only be stopped during a certain window, configure the 7 day (7), or 14 day (14) option.

We recommend that you choose a shorter waiting period in order to pick up newer platform versions revisions sooner.

Configure the wait period by running `put-account-setting-default` or `put-account-setting` as the root user or an administrative user. Use the `fargateTaskRetirementWaitPeriod` option for the `name` and the `value` option set to one of the following values:

- `0` - AWS sends the notification, and immediately starts to retire the affected tasks.
- 7 - AWS sends the notification, and waits 7 calendar days before starting to retire the affected tasks.
- 14 - AWS sends the notification, and waits 14 calendar days before starting to retire the affected tasks.

The default is 7 days.

For more information, see, put-account-setting-default and put-account-setting in the *Amazon Elastic Container Service API Reference*.

You can run the following command to set the wait period to 14 days.

```
aws ecs put-account-setting-default --name fargateTaskRetirementWaitPeriod --value 14
```

Example output

```
{
    "setting": {
        "name": "fargateTaskRetirementWaitPeriod",
        "value": "14",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type: user"
    }
}
```

You can run `list-account-settings` to view the current Fargate task retirement wait time. Use the `effective-settings` option.

```
aws ecs list-account-settings --effective-settings
```

Increase Linux container instance network interfaces

Each Amazon ECS task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the container instance that hosts it. There is a default limit to the number of network interfaces that can be attached to an Amazon EC2 instance, and the primary network interface counts as one. For example, by default a `c5.large` instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs to the instance. Because each task using the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type.

Amazon ECS supports launching container instances with increased ENI density using supported Amazon EC2 instance types. When you use these instance types and turn on the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks on each container instance.

For example, a `c5.large` instance with `awsvpcTrunking` has an increased ENI limit of twelve. The container instance will have the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the container instance. So this configuration allows you to launch ten tasks on the container instance instead of the current two tasks.

Runtime Monitoring (Amazon GuardDuty integration)

Runtime Monitoring is an intelligent threat detection service that protects workloads running on Fargate and EC2 container instances by continuously monitoring AWS log and networking activity to identify malicious or unauthorized behavior.

The `guardDutyActivate` parameter is read-only in Amazon ECS and indicates whether Runtime Monitoring is turned on or off by your security administrator in your Amazon ECS account. GuardDuty controls this account setting on your behalf. For more information, see Protecting Amazon ECS workloads with Runtime Monitoring.

You can run `list-account-settings` to view the current GuardDuty integration setting.

```
aws ecs list-account-settings
```

Example output

```
{
    "setting": {
        "name": "guardDutyActivate",
        "value": "on",
        "principalArn": "arn:aws:iam::123456789012:doej",
        "type": aws-managed"
    }
}
```

Dual stack IPv6 VPC

Amazon ECS supports providing tasks with an IPv6 address in addition to the primary private IPv4 address.

For tasks to receive an IPv6 address, the task must use the `awsvpc` network mode, must be launched in a VPC configured for dual-stack mode, and the `dualStackIPv6` account setting must be enabled. For more information about other requirements, see Using a VPC in dual-stack mode for the EC2 launch type and Using a VPC in dual-stack mode for the Fargate launch type.

> **Important**
>
> The `dualStackIPv6` account setting can only be changed using either the Amazon ECS API or the AWS CLI. For more information, see Modifying Amazon ECS account settings.

If you had a running task using the `awsvpc` network mode in an IPv6 enabled subnet between the dates of October 1, 2020 and November 2, 2020, the default `dualStackIPv6` account setting in the Region that the task was running in is `disabled`. If that condition isn't met, the default `dualStackIPv6` setting in the Region is `enabled`.

The default is `disabled`.

Default log driver mode

Amazon ECS supports setting a default delivery mode of log messages from a container to the chosen log driver. The delivery mode affects application stability when the flow of logs from the container to the log driver is interrupted.

The `defaultLogDriverMode` setting supports two values: `blocking` and `non-blocking`. For more information about these delivery modes, see LogConfiguration in the *Amazon Elastic Container Service API Reference*.

If you don't specify a delivery mode in your container definition's `logConfiguration`, the mode you specify using this account setting will be used as the default.

The default delivery mode is `non-blocking`.

Note

On June 25, 2025, Amazon ECS changed the default log driver mode from `blocking` to `non-blocking` to prioritize task availability over logging. To continue using the `blocking` mode after this change, do one of the following:

- Set the `mode` option in your container definition's `logConfiguration` as `blocking`.
- Set the `defaultLogDriverMode` account setting to `blocking`.

To set a default log driver mode to `blocking`, you can run the following command.

```
aws ecs put-account-setting-default --name defaultLogDriverMode --value "blocking"
```

Viewing Amazon ECS account settings using the console

View your account settings in the console to see which features you have access to.

Important

The `dualStackIPv6`, `fargateFIPSMode` and the `fargateTaskRetirementWaitPeriod` account settings can only be viewed or changed using the AWS CLI.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.

Modifying Amazon ECS account settings

Modify your account settings to access Amazon ECS features.

The `guardDutyActivate` parameter is read-only in Amazon ECS and indicates whether Runtime Monitoring is turned on or off by your security administrator in your Amazon ECS account. GuardDuty controls this account setting on your behalf. For more information, see [Protecting Amazon ECS workloads with Runtime Monitoring](#).

> **Important**
>
> The `dualStackIPv6`, `fargateFIPSMode` and the `fargateTaskRetirementWaitPeriod` account settings can only be viewed or changed using the AWS CLI.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. (Optional) To increase or decrease the number of tasks that you can run in the awsvpc network mode for each EC2 instance, under **AWSVPC Trunking**, select **AWSVPC Trunking**.
6. (Optional) To use, or stop using CloudWatch Container Insights by default for clusters, under **CloudWatch Container Insights observability**, choose one of the following options:
   - To use Container Insights with enhanced observability, choose **Container Insights with enhanced observability**.
   - To use Container Insights, choose **Container Insights**.
   - To stop using Container Insights, choose **Turned off**.
7. (Optional) To enable or disable tagging authorization, under **Resource Tagging Authorization**, select or clear **Resource Tagging Authorization**.
8. (Optional) To configure a default log driver mode for when a log delivery mode isn't defined in a container's `logConfiguration`, under **Default log driver mode**, choose one of the following options:
   - To set the default log driver mode as `blocking`, choose **Blocking**.

- To set the default log driver mode as `non-blocking`, choose **Non-blocking**.

9. Choose **Save changes**.
10. On the confirmation screen, choose **Confirm** to save the selection.

Next steps

If you turned on Container Insights with enhanced observability or Container Insights, you can optionally update your existing clusters to use the feature. For more information, see Updating an Amazon ECS cluster.

Reverting to the default Amazon ECS account settings

You can use the AWS Management Console to revert your Amazon ECS account settings to the default.

The **Revert to account default** option is only available when your account settings are no longer the default settings.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. Choose **Revert to account default**.
6. On the confirmation screen, choose **Confirm** to save the selection.

Managing Amazon ECS account settings using the AWS CLI

You can manage your account settings using the Amazon ECS API, AWS CLI or SDKs. The `dualStackIPv6`, `fargateFIPSMode` and the `fargateTaskRetirementWaitPeriod` account settings can only be viewed or changed using those tools.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

For information about the available API actions for task definitions see [Account setting actions](#) in the *Amazon Elastic Container Service API Reference*.

Use one of the following commands to modify the default account setting for all users or roles on your account. These changes apply to the entire AWS account unless a user or role explicitly overrides these settings for themselves.

- put-account-setting-default (AWS CLI)

```
aws ecs put-account-setting-default --name serviceLongArnFormat --value enabled --
region us-east-2
```

  You can also use this command to modify other account settings. To do this, replace the name parameter with the corresponding account setting.

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name serviceLongArnFormat -Value enabled -Region us-
east-1 -Force
```

To modify the account settings for your user account (AWS CLI)

Use one of the following commands to modify the account settings for your user. If you're using these commands as the root user, changes apply to the entire AWS account unless a; user or role explicitly overrides these settings for themselves.

- put-account-setting (AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --region us-
east-1
```

  You can also use this command to modify other account settings. To do this, replace the name parameter with the corresponding account setting.

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -Force
```

To modify the account settings for a specific user or role (AWS CLI)

Use one of the following commands and specify the ARN of a user, role, or root user in the request to modify the account settings for a specific user or role.

- put-account-setting (AWS CLI)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --principal-arn arn:aws:iam::aws_account_id:user/principalName --region us-east-1
```

  You can also use this command to modify other account settings. To do this, replace the name parameter with the corresponding account setting.

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -PrincipalArn
 arn:aws:iam::aws_account_id:user/principalName -Region us-east-1 -Force
```

IAM roles for Amazon ECS

An IAM role is an IAM identity that you can create in your account that has specific permissions. In Amazon ECS, you can create roles to grant permissions to Amazon ECS resource such as containers or services.

The roles Amazon ECS requires depend on the task definition launch type and the features that you use. Use the following table to determine which IAM roles you need for Amazon ECS.

Role	Definition	When required	More information
Task execution role	This role allows Amazon ECS to use other AWS services on your behalf.	Your task is hosted on AWS Fargate or on external instances and: • pulls a container image from an Amazon ECR private repository.	Amazon ECS task execution IAM role

Role	Definition	When required	More information
		• pulls a container image from an Amazon ECR private repository in a different account from the account that runs the task. • sends container logs to CloudWatch Logs using the `awslogs` log driver. Your task is hosted on either AWS Fargate or Amazon EC2 instances and: • uses private registry authentication. • uses Runtime Monitoring. • the task definition references sensitive data using Secrets Manager secrets or AWS Systems Manager Parameter Store parameters.	

Role	Definition	When required	More information
Task role	This role allows your application code (on the container) to use other AWS services.	Your application accesses other AWS services, such as Amazon S3.	Amazon ECS task IAM role
Container instance role	This role allows your EC2 instances or external instances to register with the cluster.	Your task is hosted on Amazon EC2 instances or an external instance.	Amazon ECS container instance IAM role
Amazon ECS Anywhere role	This role allows your external instances to access AWS APIs.	Your task is hosted on external instances.	Amazon ECS Anywhere IAM role
Amazon ECS infrastructure for load balancers role	This role allows Amazon ECS to manage load balancer resources in your clusters on your behalf for blue/green deployments.	You want to use Amazon ECS blue/green deployments.	Amazon ECS infrastructure IAM role for load balancers
Amazon ECS CodeDeploy role	This role allows CodeDeploy to make updates to your services.	You use the CodeDeploy blue/green deployment type to deploy services.	Amazon ECS CodeDeploy IAM Role
Amazon ECS EventBridge role	This role allows EventBridge to make updates to your services.	You use the EventBridge rules and targets to schedule your tasks.	Amazon ECS EventBridge IAM Role

Role	Definition	When required	More information
Amazon ECS infrastructure role	This role allows Amazon ECS to manage infrastructure resources in your clusters.	• You want to attach Amazon EBS volumes to your Fargate or EC2 launch type Amazon ECS tasks. The infrastructure role allows Amazon ECS to manage Amazon EBS volumes for your tasks. • You want to use Transport Layer Security (TLS) to encrypt traffic between your Amazon ECS Service Connect services. • You want to create VPC Lattice target groups.	Amazon ECS infrastructure IAM role

Amazon ECS task definitions

A *task definition* is a blueprint for your application. It is a text file in JSON format that describes the parameters and one or more containers that form your application.

The following are some of the parameters that you can specify in a task definition:

- The launch type to use, which determines the infrastructure that your tasks are hosted on
- The Docker image to use with each container in your task
- How much CPU and memory to use with each task or each container within a task
- The memory and CPU requirements
- The operating system of the container that the task runs on
- The Docker networking mode to use for the containers in your task
- The logging configuration to use for your tasks
- Whether the task continues to run if the container finishes or fails
- The command that the container runs when it's started
- Any data volumes that are used with the containers in the task
- The IAM role that your tasks use

For a complete list of task definition parameters, see Amazon ECS task definition parameters for the Fargate launch type.

After you create a task definition, you can run the task definition as a task or a service.

- A *task* is the instantiation of a task definition within a cluster. After you create a task definition for your application within Amazon ECS, you can specify the number of tasks to run on your cluster.
- An Amazon ECS *service* runs and maintains your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

Topics

- [Architect your application for Amazon ECS]
- [Creating an Amazon ECS task definition using the console]
- [Updating an Amazon ECS task definition using the console]
- [Deregistering an Amazon ECS task definition revision using the console]
- [Deleting an Amazon ECS task definition revision using the console]
- [Amazon ECS task definition use cases]
- [Amazon ECS task definition parameters for the Fargate launch type]
- [Amazon ECS task definition parameters for the EC2 launch type]
- [Amazon ECS task definition template]
- [Example Amazon ECS task definitions]

Amazon ECS task definition states

A task definition changes states when you create, deregister, or delete it. You can view the task definition state in the console, or by using `DescribeTaskDefinition`.

The following are the possible states for a task definition:

ACTIVE

A task definition is `ACTIVE` after it is registered with Amazon ECS. You can use task definitions in the `ACTIVE` state to run tasks, or create services.

INACTIVE

A task definition transitions from the `ACTIVE` state to the `INACTIVE` state when you deregister a task definition. You can retrieve an `INACTIVE` task definition by calling `DescribeTaskDefinition`. You cannot run new tasks or create new services with a task definition in the `INACTIVE` state. There is no impact on existing services or tasks.

DELETE_IN_PROGRESS

A task definition transitions from the `INACTIVE` state to the `DELETE_IN_PROGRESS` state after you submitted the task definition for deletion. After the task definition is in the `DELETE_IN_PROGRESS` state, Amazon ECS periodically verifies that the target task definition is not being referenced by any active tasks or deployments, and then deletes the task definition permanently. You cannot run new tasks or create new services with a task definition in the

DELETE_IN_PROGRESS state. A task definition can be submitted for deletion at any moment without impacting existing tasks and services.

Task definitions that are in the DELETE_IN_PROGRESS state can be viewed in the console and you can retrieve the task definition by calling DescribeTaskDefinition.

When you delete all INACTIVE task definition revisions, the task definition name is not displayed in the console and not returned in the API. If a task definition revision is in the DELETE_IN_PROGRESS state, the task definition name is displayed in the console and returned in the API. The task definition name is retained by Amazon ECS and the revision is incremented the next time you create a task definition with that name.

If you use AWS Config to manage your task definitions, AWS Config charges you for all task definition registrations. You are only charged for deregistering the latest ACTIVE task definition. There is no charge for deleting a task definition. For more information about pricing, see AWS Config Pricing.

Amazon ECS resources that can block a deletion

A task definition deletion request will not complete when there are any Amazon ECS resources that depend on the task definition revision. The following resources might prevent a task definition from being deleted:

- Amazon ECS standalone tasks - The task definition is required in order for the task to remain healthy.
- Amazon ECS service tasks - The task definition is required in order for the task to remain healthy.
- Amazon ECS service deployments and task sets - The task definition is required when a scaling event is initiated for an Amazon ECS deployment or task set.

If your task definition remains in the DELETE_IN_PROGRESS state, you can use the console, or the AWS CLI to identify, and then stop the resources which block the task definition deletion.

Task definition deletion after the blocked resource is removed

The following rules apply after you remove the resources that block the task definition deletion:

- Amazon ECS tasks - The task definition deletion can take up to 1 hour to complete after the task is stopped.

- Amazon ECS service deployments and task sets - The task definition deletion can take up to 24 hours to complete after the deployment or task set is deleted.

Architect your application for Amazon ECS

You architect your application by creating a task definition for your application. The task definition contains the parameters that define information about the application, including:

- The launch type to use, which determines the infrastructure that your tasks are hosted on.

  When you use the EC2 launch type, you also choose the instance type. For some instance types, such as GPU, you need to set additional parameters. For more information, see Amazon ECS task definition use cases.
- The container image, which holds your application code and all the dependencies that your application code requires to run.
- The networking mode to use for the containers in your task.

  The networking mode determines how your task communicates over the network.

  For tasks that run on EC2 instances, there are multiple options, but we recommend that you use the `awsvpc` network mode. The `awsvpc` network mode simplifies container networking by giving you more control over how your applications communicate with each other and other services within your VPCs.

  For tasks that run on Fargate, you must use the `awsvpc` network mode.
- The logging configuration to use for your tasks.
- Any data volumes that are used with the containers in the task.

For a complete list of task definition parameters, see Amazon ECS task definition parameters for the Fargate launch type.

Use the following guidelines when you create your task definitions:

- Use each task definition family for only one business purpose.

  If you group multiple types of application containers together in the same task definition, you can't independently scale those containers. For example, a website and an API typically require different scaling patterns. As traffic increases, you might need a different number of web containers than API containers. If these two containers are deployed in the same task definition, every task runs the same number of web containers and API containers.

- Match each application version with a task definition revision within a task definition family.

  Within a task definition family, each task definition revision represents a point-in-time snapshot of the settings for a particular container image. This is similar to how the container is a snapshot of all the components needed to run a particular version of your application code.

  Create a one-to-one mapping between a version of application code, a container image tag, and a task definition revision. A typical release process involves a git commit that gets turned into a container image that's tagged with the git commit SHA. Then, that container image tag gets its own Amazon ECS task definition revision. Last, the Amazon ECS service is updated to deploy the new task definition revision.
- Use different IAM roles for each task definition family.

  Define each task definition with its own IAM role. Implement this practice along with providing each business component its own task definition family. By implementing both of these best practices, you can limit how much access each service has to resources in your AWS account. For example, you can give your authentication service access to connect to your passwords database. At the same time, you can ensure that only your order service has access to the credit card payment information.

Best practices for Amazon ECS container images

A container image is a set of instructions on how to build the container. A container image holds your application code and all the dependencies that your application code requires to run. Application dependencies include the source code packages that your application code relies on, a language runtime for interpreted languages, and binary packages that your dynamically linked code relies on.

Use the following guidelines when you design and build your container images:

- Make your container images complete by storing all application dependencies as static files inside the container image.

  If you change something in the container image, build a new container image with the changes.
- Run a single application process within a container.

The container lifetime is as long as the application process runs. Amazon ECS replaces crashed processes and determines where to launch the replacement process. A complete image makes the overall deployment more resilient.

- Make your application handle `SIGTERM`.

  When Amazon ECS stops a task, it first sends a SIGTERM signal to the task to notify the application that it needs to finish and shut down. Amazon ECS then sends a `SIGKILL` message. When applications ignore the `SIGTERM`, the Amazon ECS service must wait to send the `SIGKILL` signal to terminate the process.

  You need to identify how long it takes your application to complete its work, and ensure that your applications handles the `SIGTERM` signal. The application's signal handling needs to stop the application from taking new work and complete the work that is in-progress, or save unfinished work to storage outside of the task when the work takes too long to complete.

- Configure containerized applications to write logs to `stdout` and `stderr`.

  Decoupling log handling from your application code gives you flexibility to adjust log handling at the infrastructure level. One example of this is to change your logging system. Instead of modifying your services, and building and deploying a new container image, you can adjust the settings.

- Use tags to version your container images.

  Container images are stored in a container registry. Each image in a registry is identified by a tag. There's a tag called `latest`. This tag functions as a pointer to the latest version of the application container image, similar to the `HEAD` in a git repository. We recommend that you use the `latest` tag only for testing purposes. As a best practice, tag container images with a unique tag for each build. We recommend that you tag your images using the git SHA for the git commit that was used to build the image.

  You don’t need to build a container image for every commit. However, we recommend that you build a new container image each time you release a particular code commit to the production environment. We also recommend that you tag the image with a tag that corresponds to the git commit of the code that's inside the image. If you tagged the image with the git commit, you can more quickly find which version of the code the image is running.

  We also recommend that you turn on immutable image tags in Amazon Elastic Container Registry. With this setting, you can't change the container image that a tag points at. Instead

Amazon ECR enforces that a new image must be uploaded to a new tag. For more information, see Image tag mutability in the *Amazon ECR User Guide*.

When you architect your application to run on AWS Fargate, you must decide between deploying multiple containers into the same task definition and deploying containers separately in multiple task definitions. If the following conditions are required, we recommend deploying multiple containers into the same task definition:

- Your containers share a common lifecycle (that is, they're launched and terminated together).
- Your containers must run on the same underlying host (that is, one container references the other on a localhost port).
- Your containers share resources.
- Your containers share data volumes.

If these conditions aren't required, we recommend deploying containers separately in multiple task definitions. This allows you to scale, provision, and deprovision the containers separately.

Best practices for Amazon ECS task sizes

Your container and task sizes are both essential for scaling and capacity planning. In Amazon ECS, CPU and memory are two resource metrics used for capacity. CPU is measured in units of 1/1024 of a full vCPU (where 1024 units is equal to 1 whole vCPU). Memory is measured in megabytes. In your task definition, you can configure resource reservations and limits.

When you configure a reservation, you're setting the minimum amount of resources that a task requires. Your task receives at least the amount of resources requested. Your application might be able to use more CPU or memory than the reservation that you declare. However, this is subject to any limits that you also declared. Using more than the reservation amount is known as bursting. In Amazon ECS, reservations are guaranteed. For example, if you use Amazon EC2 instances to provide capacity, Amazon ECS doesn't place a task on an instance where the reservation can't be fulfilled.

A limit is the maximum amount of CPU units or memory that your container or task can use. Any attempt to use more CPU more than this limit results in throttling. Any attempt to use more memory results in your container being stopped.

Choosing these values can be challenging. This is because the values that are the most well suited for your application greatly depend on the resource requirements of your application. Load testing your application is the key to successful resource requirement planning and better understanding your application's requirements.

Stateless applications

For stateless applications that scale horizontally, such as an application behind a load balancer, we recommend that you first determine the amount of memory that your application consumes when it serves requests. To do this, you can use traditional tools such as `ps` or `top`, or monitoring solutions such as CloudWatch Container Insights.

When determining a CPU reservation, consider how you want to scale your application to meet your business requirements. You can use smaller CPU reservations, such as 256 CPU units (or 1/4 vCPU), to scale out in a fine-grained way that minimizes cost. But, they might not scale fast enough to meet significant spikes in demand. You can use larger CPU reservations to scale in and out more quickly and therefore match demand spikes more quickly. However, larger CPU reservations are more costly.

Other applications

For applications that don't scale horizontally, such as singleton workers or database servers, available capacity and cost represent your most important considerations. You should choose the amount of memory and CPU based on what load testing indicates you need to serve traffic to meet your service-level objective. Amazon ECS ensures that the application is placed on a host that has adequate capacity.

Amazon ECS task networking options for the EC2 launch type

The networking behavior of Amazon ECS tasks that are hosted on Amazon EC2 instances is dependent on the *network mode* that's defined in the task definition. We recommend that you use the `awsvpc` network mode unless you have a specific need to use a different network mode.

The following are the available network modes.

Network mode	Linux containers on EC2	Windows containers on EC2	Description
`awsvpc`	Yes	Yes	The task is allocated its own elastic network interface (ENI) and a primary private IPv4

Network mode	Linux containers on EC2	Windows containers on EC2	Description
			address. This gives the task the same networking properties as Amazon EC2 instances.
`bridge`	Yes	No	The task uses Docker's built-in virtual network on Linux, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Linux uses the `bridge` Docker network driver. This is the default network mode on Linux if a network mode isn't specified in the task definition.
`host`	Yes	No	The task uses the host's network which bypasses Docker's built-in virtual network by mapping container ports directly to the ENI of the Amazon EC2 instance that hosts the task. Dynamic port mappings can't be used in this network mode. A container in a task definition that uses this mode must specify a specific `hostPort` number. A port number on a host can't be used by multiple tasks. As a result, you can't run multiple tasks of the same task definition on a single Amazon EC2 instance.
`none`	Yes	No	The task has no external network connectivity.
`default`	No	Yes	The task uses Docker's built-in virtual network on Windows, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Windows uses the `nat` Docker network driver. This is the default network mode on Windows if a network mode isn't specified in the task definition.

For more information about Docker networking on Linux, see Networking overview in the *Docker Documentation*.

For more information about Docker networking on Windows, see Windows container networking in the Microsoft *Containers on Windows Documentation*.

Allocate a network interface for an Amazon ECS task

The task networking features that are provided by the `awsvpc` network mode give Amazon ECS tasks the same networking properties as Amazon EC2 instances. Using the `awsvpc` network mode simplifies container networking, because you have more control over how your applications communicate with each other and other services within your VPCs. The `awsvpc` network mode also provides greater security for your containers by allowing you to use security groups and network monitoring tools at a more granular level within your tasks. You can also use other Amazon EC2 networking features such as VPC Flow Logs to monitor traffic to and from your tasks. Additionally, containers that belong to the same task can communicate over the `localhost` interface.

The task elastic network interface (ENI) is a fully managed feature of Amazon ECS. Amazon ECS creates the ENI and attaches it to the host Amazon EC2 instance with the specified security group. The task sends and receives network traffic over the ENI in the same way that Amazon EC2 instances do with their primary network interfaces. Each task ENI is assigned a private IPv4 address by default. If your VPC is enabled for dual-stack mode and you use a subnet with an IPv6 CIDR block, the task ENI will also receive an IPv6 address. Each task can only have one ENI.

These ENIs are visible in the Amazon EC2 console for your account. Your account can't detach or modify the ENIs. This is to prevent accidental deletion of an ENI that is associated with a running task. You can view the ENI attachment information for tasks in the Amazon ECS console or with the DescribeTasks API operation. When the task stops or if the service is scaled down, the task ENI is detached and deleted.

When you need increased ENI density, use the `awsvpcTrunking` account setting. Amazon ECS also creates and attaches a "trunk" network interface for your container instance. The trunk network is fully managed by Amazon ECS. The trunk ENI is deleted when you either terminate or deregister your container instance from the Amazon ECS cluster. For more information about the `awsvpcTrunking` account setting, see Prerequisites.

You specify `awsvpc` in the `networkMode` parameter of the task definition. For more information, see Network mode.

Then, when you run a task or create a service, use the `networkConfiguration` parameter that includes one or more subnets to place your tasks in and one or more security groups to attach to an ENI. For more information, see Network configuration. The tasks are placed on compatible Amazon EC2 instances in the same Availability Zones as those subnets, and the specified security groups are associated with the ENI that's provisioned for the task.

Linux considerations

Consider the following when using the Linux operating system.

- If you use a p5.48xlarge instance in `awsvpc` mode, you can't run more than 1 task on the instance.
- Tasks and services that use the `awsvpc` network mode require the Amazon ECS service-linked role to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster or if you create or update a service, in the AWS Management Console. For more information, see Using service-linked roles for Amazon ECS. You can also create the service-linked role with the following AWS CLI command:

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Your Amazon EC2 Linux instance requires version `1.15.0` or later of the container agent to run tasks that use the `awsvpc` network mode. If you're using an Amazon ECS-optimized AMI, your instance needs at least version `1.15.0-4` of the `ecs-init` package as well.
- Amazon ECS populates the hostname of the task with an Amazon-provided (internal) DNS hostname when both the `enableDnsHostnames` and `enableDnsSupport` options are enabled on your VPC. If these options aren't enabled, the DNS hostname of the task is set to a random hostname. For more information about the DNS settings for a VPC, see Using DNS with Your VPC in the *Amazon VPC User Guide*.
- Each Amazon ECS task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the Amazon EC2 instance that hosts it. There's a default quota for the number of network interfaces that can be attached to an Amazon EC2 Linux instance. The primary network interface counts as one toward that quota. For example, by default, a `c5.large` instance might have only up to three ENIs that can be attached to it. The primary network interface for the instance counts as one. You can attach an additional two ENIs to the instance. Because each task that uses the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type. For more information about the default

ENI limits for each instance type, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide*.

- Amazon ECS supports the launch of Amazon EC2 Linux instances that use supported instance types with increased ENI density. When you opt in to the `awsvpcTrunking` account setting and register Amazon EC2 Linux instances that use these instance types to your cluster, these instances have higher ENI quota. Using these instances with this higher quota means that you can place more tasks on each Amazon EC2 Linux instance. To use the increased ENI density with the trunking feature, your Amazon EC2 instance must use version `1.28.1` or later of the container agent. If you're using an Amazon ECS-optimized AMI, your instance also requires at least version `1.28.1-2` of the `ecs-init` package. For more information about opting in to the `awsvpcTrunking` account setting, see Access Amazon ECS features with account settings. For more information about ENI trunking, see Increasing Amazon ECS Linux container instance network interfaces.
- When hosting tasks that use the `awsvpc` network mode on Amazon EC2 Linux instances, your task ENIs aren't given public IP addresses. To access the internet, tasks must be launched in a private subnet that's configured to use a NAT gateway. For more information, see NAT gateways in the *Amazon VPC User Guide*. Inbound network access must be from within a VPC that uses the private IP address or routed through a load balancer from within the VPC. Tasks that are launched within public subnets do not have access to the internet.
- Amazon ECS recognizes only the ENIs that it attaches to your Amazon EC2 Linux instances. If you manually attached ENIs to your instances, Amazon ECS might attempt to add a task to an instance that doesn't have enough network adapters. This can result in the task timing out and moving to a deprovisioning status and then a stopped status. We recommend that you don't attach ENIs to your instances manually.
- Amazon EC2 Linux instances must be registered with the `ecs.capability.task-eni` capability to be considered for placement of tasks with the `awsvpc` network mode. Instances running version `1.15.0-4` or later of `ecs-init` are registered with this attribute automatically.
- The ENIs that are created and attached to your Amazon EC2 Linux instances cannot be detached manually or modified by your account. This is to prevent the accidental deletion of an ENI that is associated with a running task. To release the ENIs for a task, stop the task.
- There is a limit of 16 subnets and 5 security groups that are able to be specified in the `awsVpcConfiguration` when running a task or creating a service that uses the `awsvpc` network mode. For more information, see AwsVpcConfiguration in the *Amazon Elastic Container Service API Reference*.

- When a task is started with the `awsvpc` network mode, the Amazon ECS container agent creates an additional `pause` container for each task before starting the containers in the task definition. It then configures the network namespace of the `pause` container by running the amazon-ecs-cni-plugins CNI plugins. The agent then starts the rest of the containers in the task so that they share the network stack of the `pause` container. This means that all containers in a task are addressable by the IP addresses of the ENI, and they can communicate with each other over the `localhost` interface.
- Services with tasks that use the `awsvpc` network mode only support Application Load Balancer and Network Load Balancer. When you create any target groups for these services, you must choose `ip` as the target type. Do not use `instance`. This is because tasks that use the `awsvpc` network mode are associated with an ENI, not with an Amazon EC2 Linux instance. For more information, see Use load balancing to distribute Amazon ECS service traffic.
- If your VPC is updated to change the DHCP options set it uses, you can't apply these changes to existing tasks. Start new tasks with these changes applied to them, verify that they are working correctly, and then stop the existing tasks in order to safely change these network configurations.

Windows considerations

The following are considerations when you use the Windows operating system:

- Container instances using the Amazon ECS optimized Windows Server 2016 AMI can't host tasks that use the `awsvpc` network mode. If you have a cluster that contains Amazon ECS optimized Windows Server 2016 AMIs and Windows AMIs that support `awsvpc` network mode, tasks that use `awsvpc` network mode aren't launched on the Windows 2016 Server instances. Rather, they're launched on instances that support `awsvpc` network mode.
- Your Amazon EC2 Windows instance requires version `1.57.1` or later of the container agent to use CloudWatch metrics for Windows containers that use the `awsvpc` network mode.
- Tasks and services that use the `awsvpc` network mode require the Amazon ECS service-linked role to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster, or if you create or update a service, in the AWS Management Console. For more information, see Using service-linked roles for Amazon ECS. You can also create the service-linked role with the following AWS CLI command.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Your Amazon EC2 Windows instance requires version `1.54.0` or later of the container agent to run tasks that use the `awsvpc` network mode. When you bootstrap the instance, you must configure the options that are required for `awsvpc` network mode. For more information, see the section called "Bootstrapping container instances".
- Amazon ECS populates the hostname of the task with an Amazon provided (internal) DNS hostname when both the `enableDnsHostnames` and `enableDnsSupport` options are enabled on your VPC. If these options aren't enabled, the DNS hostname of the task is a random hostname. For more information about the DNS settings for a VPC, see Using DNS with Your VPC in the *Amazon VPC User Guide*.
- Each Amazon ECS task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the Amazon EC2 Windows instance that hosts it. There is a default quota for the number of network interfaces that can be attached to an Amazon EC2 Windows instance. The primary network interface counts as one toward this quota. For example, by default a `c5.large` instance might have only up to three ENIs attached to it. The primary network interface for the instance counts as one of those. You can attach an additional two ENIs to the instance. Because each task using the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type. For more information about the default ENI limits for each instance type, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide*.
- When hosting tasks that use the `awsvpc` network mode on Amazon EC2 Windows instances, your task ENIs aren't given public IP addresses. To access the internet, launch tasks in a private subnet that's configured to use a NAT gateway. For more information, see NAT gateways in the *Amazon VPC User Guide*. Inbound network access must be from within the VPC that is using the private IP address or routed through a load balancer from within the VPC. Tasks that are launched within public subnets don't have access to the internet.
- Amazon ECS recognizes only the ENIs that it has attached to your Amazon EC2 Windows instance. If you manually attached ENIs to your instances, Amazon ECS might attempt to add a task to an instance that doesn't have enough network adapters. This can result in the task timing out and moving to a deprovisioning status and then a stopped status. We recommend that you don't attach ENIs to your instances manually.
- Amazon EC2 Windows instances must be registered with the `ecs.capability.task-eni` capability to be considered for placement of tasks with the `awsvpc` network mode.
- You can't manually modify or detach ENIs that are created and attached to your Amazon EC2 Windows instances. This is to prevent you from accidentally deleting an ENI that's associated with a running task. To release the ENIs for a task, stop the task.

- You can only specify up to 16 subnets and 5 security groups in `awsVpcConfiguration` when you run a task or create a service that uses the `awsvpc` network mode. For more information, see [AwsVpcConfiguration](#) in the *Amazon Elastic Container Service API Reference*.
- When a task is started with the `awsvpc` network mode, the Amazon ECS container agent creates an additional `pause` container for each task before starting the containers in the task definition. It then configures the network namespace of the `pause` container by running the amazon-ecs-cni-plugins CNI plugins. The agent then starts the rest of the containers in the task so that they share the network stack of the `pause` container. This means that all containers in a task are addressable by the IP addresses of the ENI, and they can communicate with each other over the `localhost` interface.
- Services with tasks that use the `awsvpc` network mode only support Application Load Balancer and Network Load Balancer. When you create any target groups for these services, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an ENI, not with an Amazon EC2 Windows instance. For more information, see Use load balancing to distribute Amazon ECS service traffic.
- If your VPC is updated to change the DHCP options set it uses, you can't apply these changes to existing tasks. Start new tasks with these changes applied to them, verify that they are working correctly, and then stop the existing tasks in order to safely change these network configurations.
- The following are not supported when you use `awsvpc` network mode in an EC2 Windows configuration:
  - Dual-stack configuration
  - IPv6
  - ENI trunking

Using a VPC in dual-stack mode

When using a VPC in dual-stack mode, your tasks can communicate over IPv4, or IPv6, or both. IPv4 and IPv6 addresses are independent of each other. Therefore you must configure routing and security in your VPC separately for IPv4 and IPv6. For more information about how to configure your VPC for dual-stack mode, see Migrating to IPv6 in the *Amazon VPC User Guide*.

If you configured your VPC with an internet gateway or an outbound-only internet gateway, you can use your VPC in dual-stack mode. By doing this, tasks that are assigned an IPv6 address can access the internet through an internet gateway or an egress-only internet gateway. NAT gateways are optional. For more information, see Internet gateways and Egress-only internet gateways in the *Amazon VPC User Guide*.

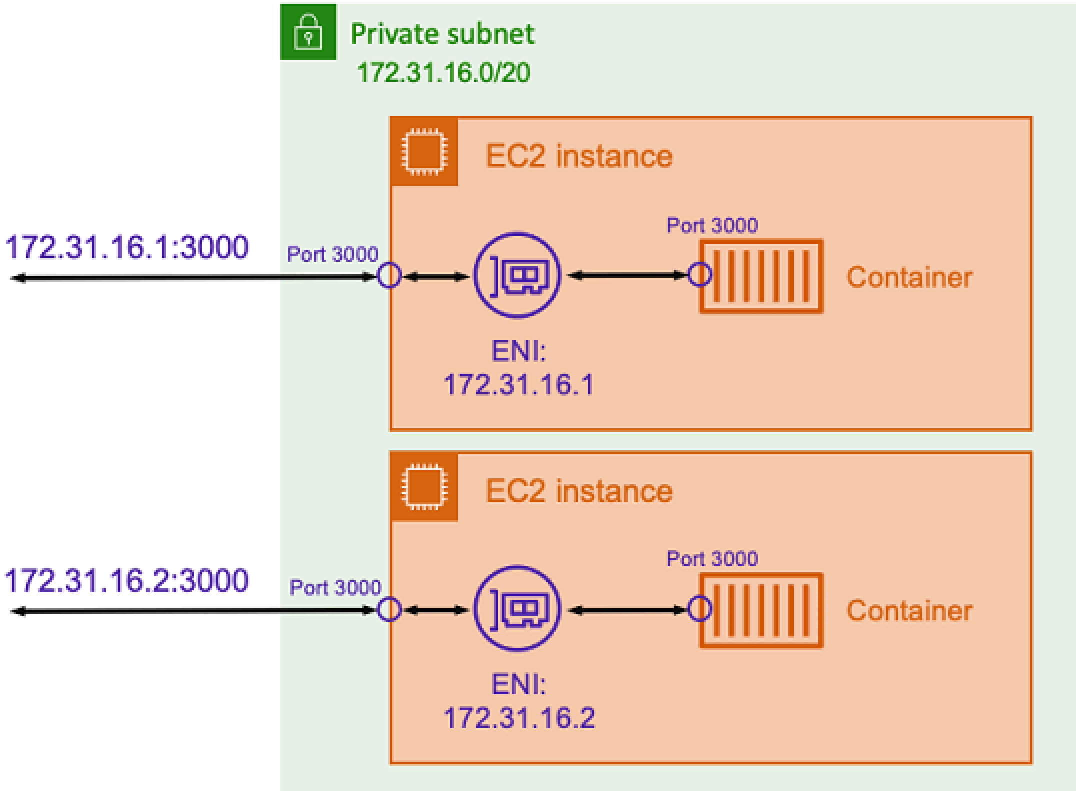
Amazon ECS tasks are assigned an IPv6 address if the following conditions are met:

- The Amazon EC2 Linux instance that hosts the task is using version `1.45.0` or later of the container agent. For information about how to check the agent version your instance is using, and updating it if needed, see Updating the Amazon ECS container agent.
- The `dualStackIPv6` account setting is enabled. For more information, see Access Amazon ECS features with account settings.
- Your task is using the `awsvpc` network mode.
- Your VPC and subnet are configured for IPv6. The configuration includes the network interfaces that are created in the specified subnet. For more information about how to configure your VPC for dual-stack mode, see Migrating to IPv6 and Modify the IPv6 addressing attribute for your subnet in the *Amazon VPC User Guide*.

Map Amazon ECS container ports to the EC2 instance network interface

The `host` network mode is only supported for Amazon ECS tasks hosted on Amazon EC2 instances. It's not supported when using Amazon ECS on Fargate.

The `host` network mode is the most basic network mode that's supported in Amazon ECS. Using host mode, the networking of the container is tied directly to the underlying host that's running the container.



Assume that you're running a Node.js container with an Express application that listens on port `3000` similar to the one illustrated in the preceding diagram. When the `host` network mode is used, the container receives traffic on port 3000 using the IP address of the underlying host Amazon EC2 instance. We do not recommend using this mode.

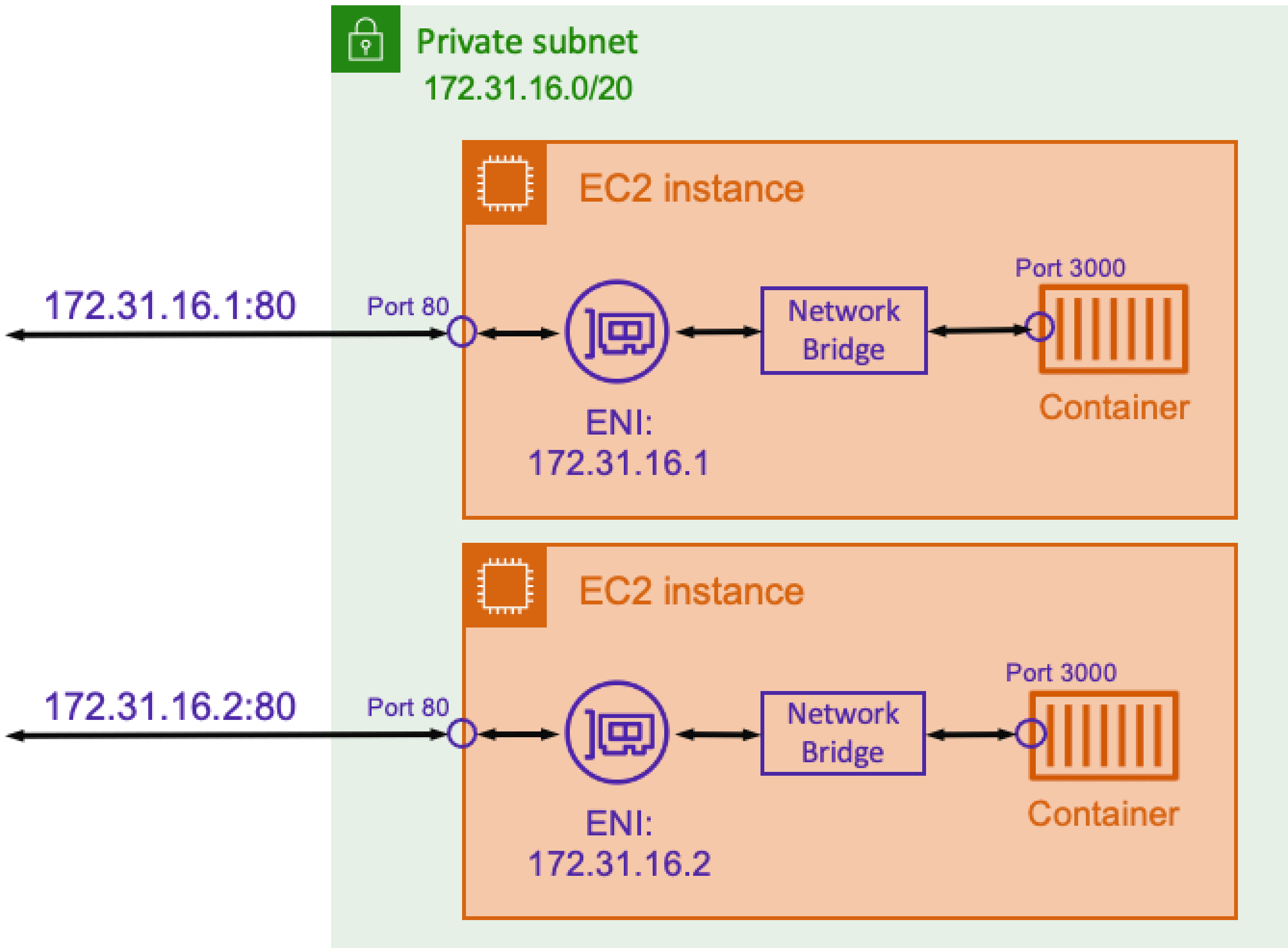
There are significant drawbacks to using this network mode. You can’t run more than a single instantiation of a task on each host. This is because only the first task can bind to its required port on the Amazon EC2 instance. There's also no way to remap a container port when it's using `host` network mode. For example, if an application needs to listen on a particular port number, you can't remap the port number directly. Instead, you must manage any port conflicts through changing the application configuration.

There are also security implications when using the `host` network mode. This mode allows containers to impersonate the host, and it allows containers to connect to private loopback network services on the host.

Use Docker's virtual network for Amazon ECS Linux tasks

The `bridge` network mode is only supported for Amazon ECS tasks hosted on Amazon EC2 instances.

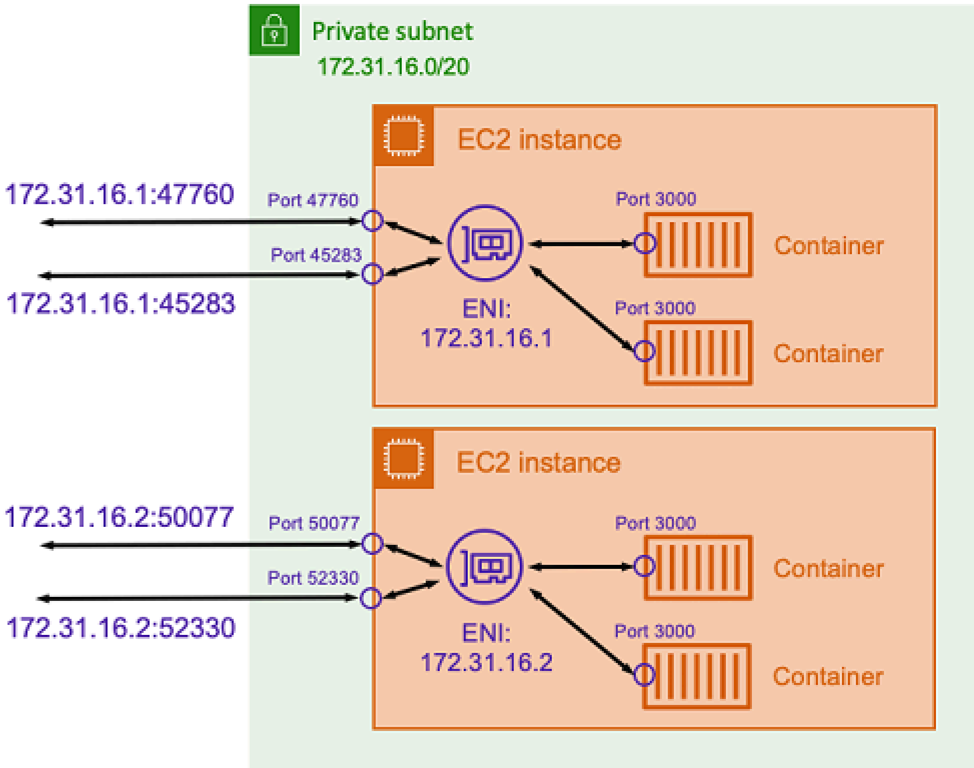
With `bridge` mode, you're using a virtual network bridge to create a layer between the host and the networking of the container. This way, you can create port mappings that remap a host port to a container port. The mappings can be either static or dynamic.



With a static port mapping, you can explicitly define which host port you want to map to a container port. Using the example above, port `80` on the host is being mapped to port `3000` on the container. To communicate to the containerized application, you send traffic to port `80` to the Amazon EC2 instance's IP address. From the containerized application's perspective it sees that inbound traffic on port `3000`.

If you only want to change the traffic port, then static port mappings is suitable. However, this still has the same disadvantage as using the `host` network mode. You can't run more than a single instantiation of a task on each host. This is because a static port mapping only allows a single container to be mapped to port 80.

To solve this problem, consider using the `bridge` network mode with a dynamic port mapping as shown in the following diagram.



By not specifying a host port in the port mapping, you can have Docker choose a random, unused port from the ephemeral port range and assign it as the public host port for the container. For example, the Node.js application listening on port `3000` on the container might be assigned a random high number port such as `47760` on the Amazon EC2 host. Doing this means that you can run multiple copies of that container on the host. Moreover, each container can be assigned its own port on the host. Each copy of the container receives traffic on port `3000`. However, clients that send traffic to these containers use the randomly assigned host ports.

Amazon ECS helps you to keep track of the randomly assigned ports for each task. It does this by automatically updating load balancer target groups and AWS Cloud Map service discovery to have the list of task IP addresses and ports. This makes it easier to use services operating using `bridge` mode with dynamic ports.

However, one disadvantage of using the `bridge` network mode is that it's difficult to lock down service to service communications. Because services might be assigned to any random, unused port, it's necessary to open broad port ranges between hosts. However, it's not easy to create specific rules so that a particular service can only communicate to one other specific service. The services have no specific ports to use for security group networking rules.

Amazon ECS task networking options for the Fargate launch type

By default, every Amazon ECS task on Fargate is provided an elastic network interface (ENI) with a primary private IP address. When using a public subnet, you can optionally assign a public IP address to the task's ENI. If your VPC is configured for dual-stack mode and you use a subnet with an IPv6 CIDR block, your task's ENI also receives an IPv6 address. A task can only have one ENI that's associated with it at a time. Containers that belong to the same task can also communicate over the `localhost` interface. For more information about VPCs and subnets, see How Amazon VPC works in the *Amazon VPC User Guide*.

For a task on Fargate to pull a container image, the task must have a route to the internet. The following describes how you can verify that your task has a route to the internet.

- When using a public subnet, you can assign a public IP address to the task ENI.
- When using a private subnet, the subnet can have a NAT gateway attached.
- When using container images that are hosted in Amazon ECR, you can configure Amazon ECR to use an interface VPC endpoint and the image pull occurs over the task's private IPv4 address. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.

Because each task gets its own ENI, you can use networking features such as VPC Flow Logs, which you can use to monitor traffic to and from your tasks. For more information, see VPC Flow Logs in the *Amazon VPC User Guide*.

You can also take advantage of AWS PrivateLink. You can configure a VPC interface endpoint so that you can access Amazon ECS APIs through private IP addresses. AWS PrivateLink restricts all network traffic between your VPC and Amazon ECS to the Amazon network. You don't need an internet gateway, a NAT device, or a virtual private gateway. For more information, see Amazon ECS interface VPC endpoints (AWS PrivateLink).

For examples of how to use the `NetworkConfiguration` resource with AWS CloudFormation, see the section called “Example AWS CloudFormation templates”.

The ENIs that are created are fully managed by AWS Fargate. Moreover, there's an associated IAM policy that's used to grant permissions for Fargate. For tasks using Fargate platform version `1.4.0` or later, the task receives a single ENI (referred to as the task ENI) and all network traffic flows through that ENI within your VPC. This traffic is recorded in your VPC flow logs. For tasks that use Fargate platform version `1.3.0` and earlier, in addition to the task ENI, the task also receives a separate Fargate owned ENI, which is used for some network traffic that isn't visible in the VPC flow logs. The following table describes the network traffic behavior and the required IAM policy for each platform version.

Action	Traffic flow with Linux platform version `1.3.0` and earlier	Traffic flow with Linux platform version `1.4.0`	Traffic flow with Windows platform version `1.0.0`	IAM permission
Retrieving Amazon ECR login credentials	Fargate owned ENI	Task ENI	Task ENI	Task execution IAM role
Image pull	Task ENI	Task ENI	Task ENI	Task execution IAM role
Sending logs through a log driver	Task ENI	Task ENI	Task ENI	Task execution IAM role
Sending logs through FireLens for Amazon ECS	Task ENI	Task ENI	Task ENI	Task IAM role
Retrieving secrets from Secrets Manager or Systems Manager	Fargate owned ENI	Task ENI	Task ENI	Task execution IAM role
Amazon EFS file system traffic	Not available	Task ENI	Task ENI	Task IAM role

Action	Traffic flow with Linux platform version `1.3.0` and earlier	Traffic flow with Linux platform version `1.4.0`	Traffic flow with Windows platform version `1.0.0`	IAM permission
Application traffic	Task ENI	Task ENI	Task ENI	Task IAM role

Considerations

Consider the following when using task networking.

- The Amazon ECS service-linked role is required to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you when you create a cluster or if you create or update a service in the AWS Management Console. For more information, see Using service-linked roles for Amazon ECS. You can also create the service-linked role using the following AWS CLI command.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Amazon ECS populates the hostname of the task with an Amazon provided DNS hostname when both the `enableDnsHostnames` and `enableDnsSupport` options are enabled on your VPC. If these options aren't enabled, the DNS hostname of the task is set to a random hostname. For more information about the DNS settings for a VPC, see Using DNS with Your VPC in the *Amazon VPC User Guide*.
- You can only specify up to 16 subnets and 5 security groups for `awsVpcConfiguration`. For more information, see AwsVpcConfiguration in the *Amazon Elastic Container Service API Reference*.
- You can't manually detach or modify the ENIs that are created and attached by Fargate. This is to prevent the accidental deletion of an ENI that's associated with a running task. To release the ENIs for a task, stop the task.
- If a VPC subnet is updated to change the DHCP options set it uses, you can't also apply these changes to existing tasks that use the VPC. Start new tasks, which will receive the new setting to smoothly migrate while testing the new change and then stop the old ones, if no rollback is required.

- The following applies to tasks run on Fargate platform version `1.4.0` or later for Linux or `1.0.0` for Windows. Tasks launched in dual-stack subnets receive an IPv4 address and an IPv6 address.
- For tasks that use platform version `1.4.0` or later for Linux or `1.0.0` for Windows, the task ENIs support jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames reduces overhead when the network path between your task and the destination supports jumbo frames.
- Services with tasks that use the Fargate launch type only support Application Load Balancer and Network Load Balancer. Classic Load Balancer isn't supported. When you create any target groups, you must choose `ip` as the target type, not `instance`. For more information, see Use load balancing to distribute Amazon ECS service traffic.

Using a VPC in dual-stack mode

When using a VPC in dual-stack mode, your tasks can communicate over IPv4 or IPv6, or both. IPv4 and IPv6 addresses are independent of each other and you must configure routing and security in your VPC separately for IPv4 and IPv6. For more information about configuring your VPC for dual-stack mode, see Migrating to IPv6 in the *Amazon VPC User Guide*.

If the following conditions are met, Amazon ECS tasks on Fargate are assigned an IPv6 address:

- Your Amazon ECS `dualStackIPv6` account setting is turned on (`enabled`) for the IAM principal launching your tasks in the Region you're launching your tasks in. This setting can only be modified using the API or AWS CLI. You have the option to turn this setting on for a specific IAM principal on your account or for your entire account by setting your account default setting. For more information, see Access Amazon ECS features with account settings.
- Your VPC and subnet are enabled for IPv6. For more information about how to configure your VPC for dual-stack mode, see Migrating to IPv6 in the *Amazon VPC User Guide*.
- Your subnet is enabled for auto-assigning IPv6 addresses. For more information about how to configure your subnet, see Modify the IPv6 addressing attribute for your subnet in the *Amazon VPC User Guide*.
- The task or service uses Fargate platform version `1.4.0` or later for Linux.

If you configure your VPC with an internet gateway or an outbound-only internet gateway, Amazon ECS tasks on Fargate that are assigned an IPv6 address can access the internet. NAT gateways aren't needed. For more information, see Internet gateways and Egress-only internet gateways in the *Amazon VPC User Guide*.

Storage options for Amazon ECS tasks

Amazon ECS provides you with flexible, cost effective, and easy-to-use data storage options depending on your needs. Amazon ECS supports the following data volume options for containers:

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
Amazon Elastic Block Store (Amazon EBS)	Fargate, Amazon EC2	Linux	Can be persisted when attached to a standalone task. Ephemeral when attached to a task maintained by a service.	Amazon EBS volumes provide cost-effective, durable, high-performance block storage for data-inte nsive container ized workloads . Common use cases include transactional workloads such as databases, virtual desktops and root volumes, and throughpu t intensive workloads such as log processin g and ETL workloads. For more informati on, see Use

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
				Amazon EBS volumes with Amazon ECS.

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
Amazon Elastic File System (Amazon EFS)	Fargate, Amazon EC2	Linux	Persistent	Amazon EFS volumes provide simple, scalable, and persisten t shared file storage for use with your Amazon ECS tasks that grows and shrinks automatically as you add and remove files. Amazon EFS volumes support concurrency and are useful for container ized applicati ons that scale horizontally and need storage functionalities like low latency, high throughpu t, and read-afte r-write consisten cy. Common use cases include workloads such as data analytics,

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
				media processing, content management, and web serving. For more information, see Use Amazon EFS volumes with Amazon ECS.

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
Amazon FSx for Windows File Server	Amazon EC2	Windows	Persistent	FSx for Windows File Server volumes provide fully managed Windows file servers that you can use to provision your Windows tasks that need persistent, distributed, shared, and static file storage. Common use cases include .NET applications that might require local folders as persistent storage to save application outputs. Amazon FSx for Windows File Server offers a local folder in the container which allows for multiple

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
				containers to read-write on the same file system that's backed by a SMB Share. For more information, see Use FSx for Windows File Server volumes with Amazon ECS.

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
Amazon FSx for NetApp ONTAP	Amazon EC2	Linux	Persistent	Amazon FSx for NetApp ONTAP volumes provide fully managed NetApp ONTAP file systems that you can use to provision your Linux tasks that need persistent, high-performance, and feature-rich shared file storage. Amazon FSx for NetApp ONTAP supports NFS and SMB protocols and provides enterprise-grade features like snapshots, cloning, and data deduplication. Common use cases include high-performance computing workloads, content

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
				repositories, and applications requiring POSIX-compliant shared storage. For more information, see Mounting Amazon FSx for NetApp ONTAP file systems from Amazon ECS containers.

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
Docker volumes	Amazon EC2	Windows, Linux	Persistent	Docker volumes are a feature of the Docker container runtime that allow container s to persist data by mounting a directory from the file system of the host. Docker volume drivers (also referred to as plugins) are used to integrate container volumes with external storage systems. Docker volumes can be managed by third-par ty drivers or by the built in `local` driver. Common use cases for Docker volumes include providing persistent data volumes or

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
				sharing volumes at different locations on different containers on the same container instance. For more informati on, see Use Docker volumes with Amazon ECS.

Data volume	Supported launch types	Supported operating systems	Storage persistence	Use cases
Bind mounts	Fargate, Amazon EC2	Windows, Linux	Ephemeral	Bind mounts consist of a file or directory on the host, such as an Amazon EC2 instance or AWS Fargate, that is mounted onto a container . Common use cases for bind mounts include sharing a volume from a source container with other containers in the same task, or mounting a host volume or an empty volume in one or more containers. For more informati on, see Use bind mounts with Amazon ECS.

Use Amazon EBS volumes with Amazon ECS

Amazon Elastic Block Store (Amazon EBS) volumes provide highly available, cost-effective, durable, high-performance block storage for data-intensive workloads. Amazon EBS volumes can be used with Amazon ECS tasks for high throughput and transaction-intensive applications.

During standalone task launch, you can provide the configuration that will be used to attach one EBS volume to the task. During service creation or update, you can provide the configuration that will be used to attach one EBS volume per task to each task managed by the Amazon ECS service.

By providing volume configuration at launch time rather than in the task definition, you create task definitions that aren't constrained to a specific data volume type or specific EBS volume settings. You can then reuse your task definitions across different runtime environments. For example, you can provide more throughput during deployment for your production workloads than your pre-prod environments.

Amazon EBS volumes that are attached to Amazon ECS tasks are managed by Amazon ECS on your behalf. The volumes can be encrypted with AWS Key Management Service (AWS KMS) keys to protect your data. You can either configure new, empty volumes for attachment, or you can use snapshots to load data from existing volumes. When you use snapshots to configure volumes, you can specify a `volumeInitializationRate`, in MiB/s, at which data is fetched from the snapshot to create volumes that are fully initialized in a predictable amount of time. For more information about volume initialization, see Initialize Amazon EBS volumes in the *Amazon EBS User Guide*. For more information about configuring Amazon EBS volumes, see Defer volume configuration to launch time in an Amazon ECS task definition and Specify Amazon EBS volume configuration at Amazon ECS deployment.

To monitor your volume's performance, you can also use Amazon CloudWatch metrics. For more information about Amazon ECS metrics for Amazon EBS volumes, see Amazon ECS CloudWatch metrics and Amazon ECS Container Insights metrics.

Attaching an Amazon EBS volume to a task is supported in all commercial and China AWS Regions that support Amazon ECS.

For more information about Amazon EBS volumes, see Amazon EBS volumes in the *Amazon EBS User Guide*.

Supported operating systems and launch types

The following table provides the supported operating system and launch type configurations.

Launch type	Linux	Windows
Fargate	Amazon EBS volumes are supported on platform version `1.4.0` or later (Linux). For more information, see Fargate platform versions for Amazon ECS.	Not supported
EC2	Tasks hosted on Nitro-based instances with Amazon ECS-optimized Amazon Machine Images (AMIs). For more information about instance types, see Instance types in the *Amazon EC2 User Guide*. Amazon EBS volumes are supported on ECS-optim ized AMI `20231219` or later. For more information, see Retrieving Amazon ECS-Optimized AMI metadata.	Tasks hosted on Nitro-based instances with Amazon ECS-optimized Amazon Machine Images (AMIs). For more information about instance types, see Instance types in the *Amazon EC2 User Guide*. Amazon EBS volumes are supported on ECS-optim ized AMI `20241017` or later. For more information, see Retrieving Amazon ECS-Optimized Windows AMI metadata.

Considerations

Consider the following when using Amazon EBS volumes:

- You can't configure Amazon EBS volumes for attachment to Fargate launch type Amazon ECS tasks in the `use1-az3` Availability Zone.
- The magnetic (`standard`) Amazon EBS volume type is not supported for tasks hosted on Fargate. For more information about Amazon EBS volume types, see Amazon EBS volumes in the *Amazon EC2 User Guide*.
- An Amazon ECS infrastructure IAM role is required when creating a service or a standalone task that is configuring a volume at deployment. You can attach the AWS managed

`AmazonECSInfrastructureRolePolicyForVolumes` IAM policy to the role, or you can use the managed policy as a guide to create and attach your own policy with permissions that meet your specific needs. For more information, see Amazon ECS infrastructure IAM role.

- You can attach at most one Amazon EBS volume to each Amazon ECS task, and it must be a new volume. You can't attach an existing Amazon EBS volume to a task. However, you can configure a new Amazon EBS volume at deployment using the snapshot of an existing volume.
- To use Amazon EBS volumes with Amazon ECS services, the deployment controller must be 'ECS'. Both rolling and blue/green deployment strategies are supported when using this deployment controller.
- For a container in your task to write to the mounted Amazon EBS volume, you must run the container as a root user.
- Amazon ECS automatically adds the reserved tags `AmazonECSCreated` and `AmazonECSManaged` to the attached volume. If you remove these tags from the volume, Amazon ECS won't be able to manage the volume on your behalf. For more information about tagging Amazon EBS volumes, see Tagging Amazon EBS volumes. For more information about tagging Amazon ECS resources, see Tagging your Amazon ECS resources.
- Provisioning volumes from a snapshot of an Amazon EBS volume that contains partitions isn't supported.
- Volumes that are attached to tasks that are managed by a service aren't preserved and are always deleted upon task termination.
- You can't configure Amazon EBS volumes for attachment to Amazon ECS tasks that are running on AWS Outposts.

Defer volume configuration to launch time in an Amazon ECS task definition

To configure an Amazon EBS volume for attachment to your task, you must specify the mount point configuration in your task definition and name the volume. You must also set `configuredAtLaunch` to `true` because Amazon EBS volumes can't be configured for attachment in the task definition. Instead, Amazon EBS volumes are configured for attachment during deployment.

To register the task definition by using the AWS Command Line Interface (AWS CLI), save the template as a JSON file, and then pass the file as an input for the `register-task-definition` command.

To create and register a task definition using the AWS Management Console, see [Creating an Amazon ECS task definition using the console](#).

The following task definition shows the syntax for the `mountPoints` and `volumes` objects in the task definition. For more information about task definition parameters, see [Amazon ECS task definition parameters for the Fargate launch type](#). To use this example, replace the *user input placeholders* with your own information.

Linux

```
{
    "family": "mytaskdef",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/nginx/nginx:latest",
            "networkMode": "awsvpc",
           "portMappings": [
                {
                    "name": "nginx-80-tcp",
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp",
                    "appProtocol": "http"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEBSVolume",
                    "containerPath": "/mount/ebs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEBSVolume",
            "configuredAtLaunch": true
        }
    ],
    "requiresCompatibilities": [
        "FARGATE", "EC2"
```

```
    ],
    "cpu": "1024",
    "memory": "3072",
    "networkMode": "awsvpc"
}
```

Windows

```
{
    "family": "mytaskdef",
     "memory": "4096",
     "cpu": "2048",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["EC2"]
    "containerDefinitions": [
        {
             "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 443,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "mountPoints": [
                {
```

```
                    "sourceVolume": "myEBSVolume",
                    "containerPath": "drive:\ebs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEBSVolume",
            "configuredAtLaunch": true
        }
    ],
    "requiresCompatibilities": [
        "FARGATE", "EC2"
    ],
    "cpu": "1024",
    "memory": "3072",
    "networkMode": "awsvpc"
}
```

`mountPoints`

Type: Object array

Required: No

The mount points for the data volumes in your container. This parameter maps to `Volumes` in the create-container Docker API and the `--volume` option to docker run.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount points cannot be used across drives. You must specify mount points to attach an Amazon EBS volume directly to an Amazon ECS task.

`sourceVolume`

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

`containerPath`

Type: String

Required: Yes, when `mountPoints` are used

The path in the container where the volume will be mounted.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

For tasks that run on EC2 instances running the Windows operating system, leave the value as the default of `false`.

`name`

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens (-), and underscores (_) are allowed. This name is referenced in the `sourceVolume` parameter of the container definition `mountPoints` object.

`configuredAtLaunch`

Type: Boolean

Required: Yes, when you want to use attach an EBS volume directly to a task.

Specifies whether a volume is configurable at launch. When set to `true`, you can configure the volume when you run a standalone task, or when you create or update a service. When set to `false`, you won't be able to provide another volume configuration in the task definition. This parameter must be provided and set to `true` to configure an Amazon EBS volume for attachment to a task.

Encrypt data stored in Amazon EBS volumes attached to Amazon ECS tasks

You can use AWS Key Management Service (AWS KMS) to make and manage cryptographic keys that protect your data. Amazon EBS volumes are encrypted at rest by using AWS KMS keys. The following types of data are encrypted:

- Data stored at rest on the volume
- Disk I/O
- Snapshots created from the volume
- New volumes created from encrypted snapshots

Amazon EBS volumes that are attached to tasks can be encrypted by using either a default AWS managed key with the alias `alias/aws/ebs`, or a symmetric customer managed key specified in the volume configuration. Default AWS managed keys are unique to each AWS account per AWS Region and are created automatically. To create a symmetric customer managed key, follow the steps in Creating symmetric encryption KMS keys in the *AWS KMS Developer Guide*.

You can configure Amazon EBS encryption by default so that all new volumes created and attached to a task in a specific AWS Region are encrypted by using the KMS key that you specify for your account. For more information about Amazon EBS encryption and encryption by default, see Amazon EBS encryption in the *Amazon EBS User Guide*.

You can also set up Amazon ECS cluster-level encryption for Amazon ECS managed storage when you create or update a cluster. Cluster-level encryption can be used to encrypt all Amazon EBS volumes attached to tasks running in a specific cluster by using the KMS key specified at the cluster level. For more information about configuring encryption at the cluster level, see ManagedStorageConfiguration in the *Amazon ECS API reference*.

You can configure any combination of these keys. The order of precedence of KMS keys is as follows:

1. The KMS key specified in the volume configuration. When you specify a KMS key in the volume configuration, it overrides the Amazon EBS default and any KMS key that is specified at the cluster level.
2. The KMS key specified at the cluster level. When you specify a KMS key for cluster-level encryption of Amazon ECS managed storage, it overrides Amazon EBS default encryption but does not override any KMS key that is specified in the volume configuration.

3. Amazon EBS default encryption. Default encryption applies when you don't specify either a cluster-level KMS key or a key in the volume configuration. If you enable Amazon EBS encryption by default, the default is the KMS key you specify for encryption by default. Otherwise, the default is the AWS managed key with the alias `alias/aws/ebs`.

> **Note**
>
> If you set `encrypted` to `false` in your volume configuration, specify no cluster-level KMS key, and enable Amazon EBS encryption by default, the volume will still be encrypted with the key specified for Amazon EBS encryption by default.

Customer managed KMS key policy

To encrypt an EBS volume that's attached to your task by using a customer managed key, you must configure your KMS key policy to ensure that the IAM role that you use for volume configuration has the necessary permissions to use the key. The key policy must include the `kms:CreateGrant` and `kms:GenerateDataKey*` permissions. The `kms:ReEncryptTo` and `kms:ReEncryptFrom` permissions are necessary for encrypting volumes that are created using snapshots. If you want to configure and encrypt only new, empty volumes for attachment, you can exclude the `kms:ReEncryptTo` and `kms:ReEncryptFrom` permissions.

The following JSON snippet shows key policy statements that you can attach to your KMS key policy. Using these statements will provide access for Amazon ECS to use the key for encrypting the EBS volume. To use the example policy statements, replace the *`user input placeholders`* with your own information. As always, only configure the permissions that you need.

```
{
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": "kms:DescribeKey",
      "Resource":"*"
    },
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": [
      "kms:GenerateDataKey*",
      "kms:ReEncryptTo",
      "kms:ReEncryptFrom"
```

```
      ],
      "Resource":"*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "aws_account_id",
          "kms:ViaService": "ec2.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
          "kms:EncryptionContextKeys": "aws:ebs:id"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/ecsInfrastructureRole" },
      "Action": "kms:CreateGrant",
      "Resource":"*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "aws_account_id",
          "kms:ViaService": "ec2.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
          "kms:EncryptionContextKeys": "aws:ebs:id"
        },
        "Bool": {
          "kms:GrantIsForAWSResource": true
        }
      }
    }
```

For more information about key policies and permissions, see Key policies in AWS KMS and AWS KMS permissions in the *AWS KMS Developer Guide*. For troubleshooting EBS volume attachment issues related to key permissions, see Troubleshooting Amazon EBS volume attachments to Amazon ECS tasks .

Specify Amazon EBS volume configuration at Amazon ECS deployment

After you register a task definition with the `configuredAtLaunch` parameter set to `true`, you can configure an Amazon EBS volume at deployment when you run a standalone task, or when you create or update a service.

To configure a volume, you can use the Amazon ECS APIs, or you can pass a JSON file as input for the following AWS CLI commands:

- `run-task` to run a standalone ECS task.
- `start-task` to run a standalone ECS task in a specific container instance. This command is not applicable for Fargate launch type tasks.
- `create-service` to create a new ECS service.
- `update-service` to update an existing service.

> **Note**
>
> For a container in your task to write to the mounted Amazon EBS volume, you must run the container as a root user.

You can also configure an Amazon EBS volume by using the AWS Management Console. For more information, see Running an application as an Amazon ECS task, Creating an Amazon ECS rolling update deployment, and Updating an Amazon ECS service.

The following JSON snippet shows all the parameters of an Amazon EBS volume that can be configured at deployment. To use these parameters for volume configuration, replace the *user input placeholders* with your own information. For more information about these parameters, see Volume configurations.

```
"volumeConfigurations": [
        {
            "name": "ebs-volume",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
                "volumeType": "gp3",
                "sizeInGiB": 10,
                "snapshotId": "snap-12345",
                "volumeInitializationRate":100,
                "iops": 3000,
                "throughput": 125,
                "tagSpecifications": [
                    {
```

```
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "key1",
                                "value": "value1"
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "arn:aws:iam::1111222333:role/ecsInfrastructureRole",
                 "terminationPolicy": {
                    "deleteOnTermination": true//can't be configured for service-
managed tasks, always true
                },
                "filesystemType": "ext4"
            }
        }
    ]
```

> **Important**
>
> Ensure that the `volumeName` you specify in the configuration is the same as the `volumeName` you specify in your task definition.

For information about checking the status of volume attachment, see Troubleshooting Amazon EBS volume attachments to Amazon ECS tasks . For information about the Amazon ECS infrastructure AWS Identity and Access Management (IAM) role necessary for EBS volume attachment, see Amazon ECS infrastructure IAM role.

The following are JSON snippet examples that show the configuration of Amazon EBS volumes. These examples can be used by saving the snippets in JSON files and passing the files as parameters (using the `--cli-input-json file://filename` parameter) for AWS CLI commands. Replace the *user input placeholders* with your own information.

Configure a volume for a standalone task

The following snippet shows the syntax for configuring Amazon EBS volumes for attachment to a standalone task. The following JSON snippet shows the syntax for configuring the `volumeType`, `sizeInGiB`, `encrypted`, and `kmsKeyId` settings. The configuration specified in the JSON file is used to create and attach an EBS volume to the standalone task.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "volumeConfigurations": [
        {
            "name": "datadir",
            "managedEBSVolume": {
                "volumeType": "gp3",
                "sizeInGiB": 100,
                "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
                "encrypted": true,
                "kmsKeyId":
 "arn:aws:kms:region:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            }
        }
   ]
}
```

Configure a volume at service creation

The following snippet shows the syntax for configuring Amazon EBS volumes for attachment to tasks managed by a service. The volumes are sourced from the snapshot specified using the `snapshotId` parameter at a rate of 200 MiB/s. The configuration specified in the JSON file is used to create and attach an EBS volume to each task managed by the service.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "serviceName": "mysvc",
   "desiredCount": 2,
   "volumeConfigurations": [
        {
            "name": "myEbsVolume",
            "managedEBSVolume": {
              "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
              "snapshotId": "snap-12345",
              "volumeInitializationRate": 200
            }
        }
```

```
    ]
}
```

Configure a volume at service update

The following JSON snippet shows the syntax for updating a service that previously did not have Amazon EBS volumes configured for attachment to tasks. You must provide the ARN of a task definition revision with `configuredAtLaunch` set to `true`. The following JSON snippet shows the syntax for configuring the `volumeType`, `sizeInGiB`, `throughput`, and `iops`, and `filesystemType` settings. This configuration is used to create and attach an EBS volume to each task managed by the service.

```
{
   "cluster": "mycluster",
   "taskDefinition": "mytaskdef",
   "service": "mysvc",
   "desiredCount": 2,
   "volumeConfigurations": [
        {
            "name": "myEbsVolume",
            "managedEBSVolume": {
              "roleArn":"arn:aws:iam::1111222333:role/ecsInfrastructureRole",
               "volumeType": "gp3",
                "sizeInGiB": 100,
                 "iops": 3000,
                "throughput": 125,
                "filesystemType": "ext4"
            }
        }
   ]
}
```

Configure a service to no longer utilize Amazon EBS volumes

The following JSON snippet shows the syntax for updating a service to no longer utilize Amazon EBS volumes. You must provide the ARN of a task definition with `configuredAtLaunch` set to `false`, or a task definition without the `configuredAtLaunch` parameter. You must also provide an empty `volumeConfigurations` object.

```
{
   "cluster": "mycluster",
```

```
    "taskDefinition": "mytaskdef",
    "service": "mysvc",
    "desiredCount": 2,
    "volumeConfigurations": []
}
```

Termination policy for Amazon EBS volumes

When an Amazon ECS task terminates, Amazon ECS uses the `deleteOnTermination` value to determine whether the Amazon EBS volume that's associated with the terminated task should be deleted. By default, EBS volumes that are attached to tasks are deleted when the task is terminated. For standalone tasks, you can change this setting to instead preserve the volume upon task termination.

> **Note**
>
> Volumes that are attached to tasks that are managed by a service are not preserved and are always deleted upon task termination.

Tag Amazon EBS volumes

You can tag Amazon EBS volumes by using the `tagSpecifications` object. Using the object, you can provide your own tags and set propagation of tags from the task definition or the service, depending on whether the volume is attached to a standalone task or a task in a service. The maximum number of tags that can be attached to a volume is 50.

> **Important**
>
> Amazon ECS automatically attaches the `AmazonECSCreated` and `AmazonECSManaged` reserved tags to an Amazon EBS volume. This means you can control the attachment of a maximum of 48 additional tags to a volume. These additional tags can be user-defined, ECS-managed, or propagated tags.

If you want to add Amazon ECS-managed tags to your volume, you must set `enableECSManagedTags` to `true` in your `UpdateService`, `CreateService`,`RunTask` or `StartTask` call. If you turn on Amazon ECS-managed tags, Amazon ECS will tag the volume automatically with cluster and service information (`aws:ecs:`*`clusterName`* and `aws:ecs:serviceName`). For more information about tagging Amazon ECS resources, see [Tagging your Amazon ECS resources](#).

The following JSON snippet shows the syntax for tagging each Amazon EBS volume that is attached to each task in a service with a user-defined tag. To use this example for creating a service, replace the *`user input placeholders`* with your own information.

```
{
    "cluster": "mycluster",
    "taskDefinition": "mytaskdef",
    "serviceName": "mysvc",
    "desiredCount": 2,
    "enableECSManagedTags": true,
    "volumeConfigurations": [
         {
             "name": "datadir",
             "managedEBSVolume": {
                 "volumeType": "gp3",
                 "sizeInGiB": 100,
                  "tagSpecifications": [
                     {
                         "resourceType": "volume",
                         "tags": [
                             {
                                 "key": "key1",
                                 "value": "value1"
                             }
                         ],
                         "propagateTags": "NONE"
                     }
                 ],
                 "roleArn":"arn:aws:iam:1111222333:role/ecsInfrastructureRole",
                 "encrypted": true,
                 "kmsKeyId": 
 "arn:aws:kms:region:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
             }
         }
    ]
}
```

> **Important**
>
> You must specify a `volume` resource type to tag Amazon EBS volumes.

Performance of Amazon EBS volumes for Fargate on-demand tasks

The baseline Amazon EBS volume IOPS and throughput available for a Fargate on-demand task depends on the total CPU units you request for the task. If you request 0.25, 0.5, or 1 virtual CPU unit (vCPU) for your Fargate task, we recommend that you configure a General Purpose SSD volume (`gp2` or `gp3`) or a Hard Disk Drive (HDD) volume (`st1` or `sc1`). If you request more than 1 vCPU for your Fargate task, the following baseline performance limits apply to an Amazon EBS volume attached to the task. You may temporarily get higher EBS performance than the following limits. However, we recommend that you plan your workload based on these limits.

CPU units requested (in vCPUs)	Baseline Amazon EBS IOPS(16 KiB I/O)	Baseline Amazon EBS Throughput (in MiBps, 128 KiB I/O)	Baseline bandwidth (in Mbps)
2	3,000	75	360
4	5,000	120	1,150
8	10,000	250	2,300
16	15,000	500	4,500

> **Note**
>
> When you configure an Amazon EBS volume for attachment to a Fargate task, the Amazon EBS performance limit for the Fargate task is shared between the task's ephemeral storage and the attached volume.

Performance of Amazon EBS volumes for EC2 tasks

Amazon EBS provides volume types, which differ in performance characteristics and price, so that you can tailor your storage performance and cost to the needs of your applications. For information about performance, including IOPS per volume and throughput per volume, see Amazon EBS volume types in the *Amazon Elastic Block Store User Guide*.

Troubleshooting Amazon EBS volume attachments to Amazon ECS tasks

You might need to troubleshoot or verify the attachment of Amazon EBS volumes to Amazon ECS tasks.

Check volume attachment status

You can use the AWS Management Console to view the status of an Amazon EBS volume's attachment to an Amazon ECS task. If the task starts and the attachment fails, you'll also see a status reason that you can use to troubleshoot. The created volume will be deleted and the task will be stopped. For more information about status reasons, see Status reasons for Amazon EBS volume attachment to Amazon ECS tasks.

To view a volume's attachment status and status reason using the console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster that your task is running in. The cluster's details page appears.
3. On the cluster's details page, choose the **Tasks** tab.
4. Choose the task that you want to view the volume attachment status for. You might need to use **Filter desired status** and choose **Stopped** if the task you want to examine has stopped.
5. On the task's details page, choose the **Volumes** tab. You will be able to see the attachment status of the Amazon EBS volume under **Attachment status**. If the volume fails to attach to the task, you can choose the status under **Attachment status** to display the cause of the failure.

You can also view a task's volume attachment status and associated status reason by using the DescribeTasks API.

Service and task failures

You might encounter service or task failures that aren't specific to Amazon EBS volumes that can affect volume attachment. For more information, see

- Service event messages

- Stopped task error codes
- API failure reasons

Status reasons for Amazon EBS volume attachment to Amazon ECS tasks

Use the following reference to fix issues that you might encounter in the form of status reasons in the AWS Management Console when you configure Amazon EBS volumes for attachment to Amazon ECS tasks. For more information on locating these status reasons in the console, see Check volume attachment status.

ECS was unable to assume the configured ECS Infrastructure Role 'arn:aws:iam::*111122223333*:role/*ecsInfrastructureRole*'. Please verify that the role being passed has the proper trust relationship with Amazon ECS

This status reason appears in the following scenarios.

- You provide an IAM role without the necessary trust policy attached. Amazon ECS can't access the Amazon ECS infrastructure IAM role that you provide if the role doesn't have the necessary trust policy. The task can get stuck in the `DEPROVISIONING` state. For more information about the necessary trust policy, see Amazon ECS infrastructure IAM role.
- Your IAM user doesn't have permission to pass the Amazon ECS infrastructure role to Amazon ECS. The task can get stuck in the `DEPROVISIONING` state. To avoid this problem, you can attach the `PassRole` permission to your user. For more information, see Amazon ECS infrastructure IAM role.
- Your IAM role doesn't have the necessary permissions for Amazon EBS volume attachment. The task can get stuck in the `DEPROVISIONING` state. For more information about the specific permissions necessary for attaching Amazon EBS volumes to tasks, see Amazon ECS infrastructure IAM role.

> **Note**
>
> You may also see this error message due to a delay in role propagation. If retrying to use the role after waiting for a few minutes doesn't fix the issue, you might have misconfigured the trust policy for the role.

ECS failed to set up the EBS volume. Encountered IdempotentParameterMismatch"; "The client token you have provided is associated with a resource that is already deleted. Please use a different client token."

The following AWS KMS key scenarios can lead to an `IdempotentParameterMismatch` message appearing:

- You specify a KMS key ARN, ID, or alias that isn't valid. In this scenario, the task might appear to launch successfully, but the task eventually fails because AWS authenticates the KMS key asynchronously. For more information, see Amazon EBS encryption in the *Amazon EC2 User Guide*.
- You provide a customer managed key that lacks the permissions that allow the Amazon ECS infrastructure IAM role to use the key for encryption. To avoid key-policy permission issues, see the example AWS KMS key policy in Data encryption for Amazon EBS volumes.

You can set up Amazon EventBridge to send Amazon EBS volume events and Amazon ECS task state change events to a target, such as Amazon CloudWatch groups. You can then use these events to identify the specific customer managed key related issue that affected volume attachment. For more information, see

- How can I create a CloudWatch log group to use as a target for an EventBridge rule? on AWS re:Post.
- Task state change events.
- Amazon EventBridge events for Amazon EBS in the *Amazon EBS User Guide*.

ECS timed out while configuring the EBS volume attachment to your Task.

The following file system format scenarios result in this message.

- The file system format that you specify during configuration isn't compatible with the task's operating system.
- You configure an Amazon EBS volume to be created from a snapshot, and the snapshot's file system format isn't compatible with the task's operating system. For volumes created from a snapshot, you must specify the same filesystem type that the volume was using when the snapshot was created.

You can utilize the Amazon ECS container agent logs to troubleshoot this message for Amazon EC2 launch type tasks. For more information, see Amazon ECS log file locations and Amazon ECS log collector.

Use Amazon EFS volumes with Amazon ECS

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic. It grows and shrinks automatically as you add and remove files. Your applications can have the storage they need and when they need it.

You can use Amazon EFS file systems with Amazon ECS to export file system data across your fleet of container instances. That way, your tasks have access to the same persistent storage, no matter the instance on which they land. Your task definitions must reference volume mounts on the container instance to use the file system.

For a tutorial, see Configuring Amazon EFS file systems for Amazon ECS using the console.

Considerations

Consider the following when using Amazon EFS volumes:

- For tasks that use the EC2 launch type, Amazon EFS file system support was added as a public preview with Amazon ECS-optimized AMI version `20191212` with container agent version 1.35.0. However, Amazon EFS file system support entered general availability with Amazon ECS-optimized AMI version `20200319` with container agent version 1.38.0, which contained the Amazon EFS access point and IAM authorization features. We recommend that you use Amazon ECS-optimized AMI version `20200319` or later to use these features. For more information, see Amazon ECS-optimized Linux AMIs.

  Note

  If you create your own AMI, you must use container agent 1.38.0 or later, `ecs-init` version 1.38.0-1 or later, and run the following commands on your Amazon EC2 instance to enable the Amazon ECS volume plugin. The commands are dependent on whether you're using Amazon Linux 2 or Amazon Linux as your base image.
  Amazon Linux 2

  ```
  yum install amazon-efs-utils
  systemctl enable --now amazon-ecs-volume-plugin
  ```

  Amazon Linux

  ```
  yum install amazon-efs-utils
  ```

```
sudo shutdown -r now
```

- For tasks that are hosted on Fargate, Amazon EFS file systems are supported on platform version 1.4.0 or later (Linux). For more information, see Fargate platform versions for Amazon ECS.
- When using Amazon EFS volumes for tasks that are hosted on Fargate, Fargate creates a supervisor container that's responsible for managing the Amazon EFS volume. The supervisor container uses a small amount of the task's memory and CPU. The supervisor container is visible when querying the task metadata version 4 endpoint. Additionally, it is visible in CloudWatch Container Insights as the container name `aws-fargate-supervisor`. For more information when using the Amazon EC2 launch type, see Amazon ECS task metadata endpoint version 4. For more information when using the Fargate launch type, see Amazon ECS task metadata endpoint version 4 for tasks on Fargate.
- Using Amazon EFS volumes or specifying an `EFSVolumeConfiguration` isn't supported on external instances.
- We recommend that you set the `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` parameter in the agent configuration file to a value that is less than the default (about 1 hour). This change helps prevent EFS mount credential expiration and allows for cleanup of mounts that are not in use. For more information, see Amazon ECS container agent configuration.

Use Amazon EFS access points

Amazon EFS access points are application-specific entry points into an EFS file system for managing application access to shared datasets. For more information about Amazon EFS access points and how to control access to them, see Working with Amazon EFS Access Points in the *Amazon Elastic File System User Guide*.

Access points can enforce a user identity, including the user's POSIX groups, for all file system requests that are made through the access point. Access points can also enforce a different root directory for the file system. This is so that clients can only access data in the specified directory or its subdirectories.

> **Note**
>
> When creating an EFS access point, specify a path on the file system to serve as the root directory. When referencing the EFS file system with an access point ID in your Amazon ECS

task definition, the root directory must either be omitted or set to /, which enforces the path set on the EFS access point.

You can use an Amazon ECS task IAM role to enforce that specific applications use a specific access point. By combining IAM policies with access points, you can provide secure access to specific datasets for your applications. For more information about how to use task IAM roles, see Amazon ECS task IAM role.

Best practices for using Amazon EFS volumes with Amazon ECS

Make note of the following best practice recommendations when you use Amazon EFS with Amazon ECS.

Security and access controls for Amazon EFS volumes

Amazon EFS offers access control features that you can use to ensure that the data stored in an Amazon EFS file system is secure and accessible only from applications that need it. You can secure data by enabling encryption at rest and in-transit. For more information, see Data encryption in Amazon EFS in the *Amazon Elastic File System User Guide*.

In addition to data encryption, you can also use Amazon EFS to restrict access to a file system. There are three ways to implement access control in EFS.

- **Security groups**—With Amazon EFS mount targets, you can configure a security group that's used to permit and deny network traffic. You can configure the security group attached to Amazon EFS to permit NFS traffic (port 2049) from the security group that's attached to your Amazon ECS instances or, when using the `awsvpc` network mode, the Amazon ECS task.
- **IAM**—You can restrict access to an Amazon EFS file system using IAM. When configured, Amazon ECS tasks require an IAM role for file system access to mount an EFS file system. For more information, see Using IAM to control file system data access in the *Amazon Elastic File System User Guide*.

  IAM policies can also enforce predefined conditions such as requiring a client to use TLS when connecting to an Amazon EFS file system. For more information, see Amazon EFS condition keys for clients in the *Amazon Elastic File System User Guide*.
- **Amazon EFS access points**—Amazon EFS access points are application-specific entry points into an Amazon EFS file system. You can use access points to enforce a user identity, including the

user's POSIX groups, for all file system requests that are made through the access point. Access points can also enforce a different root directory for the file system. This is so that clients can only access data in the specified directory or its sub-directories.

IAM policies

You can use IAM policies to control the access to the Amazon EFS file system.

You can specify the following actions for clients accessing a file system using a file system policy.

Action	Description
`elasticfilesystem:ClientMount`	Provides read-only access to a file system.
`elasticfilesystem:ClientWrite`	Provides write permissions on a file system.
`elasticfilesystem:ClientRootAccess`	Provides use of the root user when accessing a file system.

You need to specify each action in a policy. The policies can be defined in the following ways:

- Client-based - Attach the policy to the task role

  Set the **IAM authorization** option when you create the task definition.
- Resource-based - Attach the policy to Amazon EFS file system

  If the resource-based policy does not exist, by default at file system creation access is granted to all principals (*).

When you set the **IAM authorization** option, we merge the the policy associated with the task role and the Amazon EFS resource-based. The **IAM authorization** option passes the task identity (the task role) with the policy to Amazon EFS. This allows the Amazon EFS resource-based policy to have context for the IAM user or role specified in the policy. If you do not set the option, the Amazon EFS resource-level policy identifies the IAM user as ”anonymous".

Consider implementing all three access controls on an Amazon EFS file system for maximum security. For example, you can configure the security group attached to an Amazon EFS mount point to only permit ingress NFS traffic from a security group that's associated with your container instance or Amazon ECS task. Additionally, you can configure Amazon EFS to require an IAM role to access the file system, even if the connection originates from a permitted security group. Last, you can use Amazon EFS access points to enforce POSIX user permissions and specify root directories for applications.

The following task definition snippet shows how to mount an Amazon EFS file system using an access point.

```
"volumes": [
    {
      "efsVolumeConfiguration": {
        "fileSystemId": "fs-1234",
        "authorizationConfig": {
          "accessPointId": "fsap-1234",
          "iam": "ENABLED"
        },
        "transitEncryption": "ENABLED",
        "rootDirectory": ""
      },
      "name": "my-filesystem"
    }
]
```

Amazon EFS volume performance

Amazon EFS offers two performance modes: General Purpose and Max I/O. General Purpose is suitable for latency-sensitive applications such as content management systems and CI/CD tools. In contrast, Max I/O file systems are suitable for workloads such as data analytics, media processing, and machine learning. These workloads need to perform parallel operations from hundreds or even thousands of containers and require the highest possible aggregate throughput and IOPS. For more information, see [Amazon EFS performance modes]() in the *Amazon Elastic File System User Guide*.

Some latency sensitive workloads require both the higher I/O levels that are provided by Max I/O performance mode and the lower latency that are provided by General Purpose performance mode. For this type of workload, we recommend creating multiple General Purpose performance mode file systems. That way, you can spread your application workload across all these file systems, as long as the workload and applications can support it.

Amazon EFS volume throughput

All Amazon EFS file systems have an associated metered throughput that's determined by either the amount of provisioned throughput for file systems using *Provisioned Throughput* or the amount of data stored in the EFS Standard or One Zone storage class for file systems using *Bursting Throughput*. For more information, see Understanding metered throughput in the *Amazon Elastic File System User Guide*.

The default throughput mode for Amazon EFS file systems is bursting mode. With bursting mode, the throughput that's available to a file system scales in or out as a file system grows. Because file-based workloads typically spike, requiring high levels of throughput for periods of time and lower levels of throughput the rest of the time, Amazon EFS is designed to burst to allow high throughput levels for periods of time. Additionally, because many workloads are read-heavy, read operations are metered at a 1:3 ratio to other NFS operations (like write).

All Amazon EFS file systems deliver a consistent baseline performance of 50 MB/s for each TB of Amazon EFS Standard or Amazon EFS One Zone storage. All file systems (regardless of size) can burst to 100 MB/s. File systems with more than 1TB of EFS Standard or EFS One Zone storage can burst to 100 MB/s for each TB. Because read operations are metered at a 1:3 ratio, you can drive up to 300 MiBs/s for each TiB of read throughput. As you add data to your file system, the maximum throughput that's available to the file system scales linearly and automatically with your storage in the Amazon EFS Standard storage class. If you need more throughput than you can achieve with your amount of data stored, you can configure Provisioned Throughput to the specific amount your workload requires.

File system throughput is shared across all Amazon EC2 instances connected to a file system. For example, a 1TB file system that can burst to 100 MB/s of throughput can drive 100 MB/s from a single Amazon EC2 instance can each drive 10 MB/s. For more information, see Amazon EFS performance in the *Amazon Elastic File System User Guide*.

Optimizing cost for Amazon EFS volumes

Amazon EFS simplifies scaling storage for you. Amazon EFS file systems grow automatically as you add more data. Especially with Amazon EFS *Bursting Throughput* mode, throughput on Amazon EFS scales as the size of your file system in the standard storage class grows. To improve the throughput without paying an additional cost for provisioned throughput on an EFS file system, you can share an Amazon EFS file system with multiple applications. Using Amazon EFS access points, you can implement storage isolation in shared Amazon EFS file systems. By doing so, even though the applications still share the same file system, they can't access data unless you authorize it.

As your data grows, Amazon EFS helps you automatically move infrequently accessed files to a lower storage class. The Amazon EFS Standard-Infrequent Access (IA) storage class reduces storage costs for files that aren't accessed every day. It does this without sacrificing the high availability, high durability, elasticity, and the POSIX file system access that Amazon EFS provides. For more information, see EFS storage classes in the *Amazon Elastic File System User Guide*.

Consider using Amazon EFS lifecycle policies to automatically save money by moving infrequently accessed files to Amazon EFS IA storage. For more information, see Amazon EFS lifecycle management in the *Amazon Elastic File System User Guide*.

When creating an Amazon EFS file system, you can choose if Amazon EFS replicates your data across multiple Availability Zones (Standard) or stores your data redundantly within a single Availability Zone. The Amazon EFS One Zone storage class can reduce storage costs by a significant margin compared to Amazon EFS Standard storage classes. Consider using Amazon EFS One Zone storage class for workloads that don't require multi-AZ resilience. You can further reduce the cost of Amazon EFS One Zone storage by moving infrequently accessed files to Amazon EFS One Zone-Infrequent Access. For more information, see Amazon EFS Infrequent Access.

Amazon EFS volume data protection

Amazon EFS stores your data redundantly across multiple Availability Zones for file systems using Standard storage classes. If you select Amazon EFS One Zone storage classes, your data is redundantly stored within a single Availability Zone. Additionally, Amazon EFS is designed to provide 99.999999999% (11 9's) of durability over a given year.

As with any environment, it's a best practice to have a backup and to build safeguards against accidental deletion. For Amazon EFS data, that best practice includes a functioning, regularly tested backup using AWS Backup. File systems using Amazon EFS One Zone storage classes are configured to automatically back up files by default at file system creation unless you choose to disable this functionality. For more information, see Backing up EFS file systems in the *Amazon Elastic File System User Guide*.

Specify an Amazon EFS file system in an Amazon ECS task definition

To use Amazon EFS file system volumes for your containers, you must specify the volume and mount point configurations in your task definition. The following task definition JSON snippet shows the syntax for the `volumes` and `mountPoints` objects for a container.

```
{
    "containerDefinitions": [
        {
            "name": "container-using-efs",
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "ls -la /mount/efs"
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEfsVolume",
                    "containerPath": "/mount/efs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEfsVolume",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1234",
                "rootDirectory": "/path/to/my/data",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": integer,
                "authorizationConfig": {
                    "accessPointId": "fsap-1234",
                    "iam": "ENABLED"
                }
            }
        }
    ]
}
```

`efsVolumeConfiguration`

Type: Object

Required: No

This parameter is specified when using Amazon EFS volumes.

`fileSystemId`

Type: String

Required: Yes

The Amazon EFS file system ID to use.

`rootDirectory`

Type: String

Required: No

The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume is used. Specifying `/` has the same effect as omitting this parameter.

> **Important**
>
> If an EFS access point is specified in the `authorizationConfig`, the root directory parameter must either be omitted or set to `/`, which enforces the path set on the EFS access point.

`transitEncryption`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. If Amazon EFS IAM authorization is used, transit encryption must be enabled. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see Encrypting Data in Transit in the *Amazon Elastic File System User Guide*.

`transitEncryptionPort`

Type: Integer

Required: No

The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you don't specify a transit encryption port, it uses the port selection strategy that the Amazon EFS mount helper uses. For more information, see EFS Mount Helper in the *Amazon Elastic File System User Guide*.

`authorizationConfig`

Type: Object

Required: No

The authorization configuration details for the Amazon EFS file system.

`accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value in the `efsVolumeConfiguration` must either be omitted or set to `/`, which enforces the path set on the EFS access point. If an access point is used, transit encryption must be enabled in the `EFSVolumeConfiguration`. For more information, see Working with Amazon EFS Access Points in the *Amazon Elastic File System User Guide*.

`iam`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the `EFSVolumeConfiguration`. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see IAM Roles for Tasks.

Configuring Amazon EFS file systems for Amazon ECS using the console

Learn how to use Amazon Elastic File System (Amazon EFS) file systems with Amazon ECS.

Step 1: Create an Amazon ECS cluster

Use the following steps to create an Amazon ECS cluster.

To create a new cluster (Amazon ECS console)

Before you begin, assign the appropriate IAM permission. For more information, see the section called “Amazon ECS cluster examples”.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, for **Cluster name**, enter `EFS-tutorial` for the cluster name.
6. (Optional) To change the VPC and subnets where your tasks and services launch, under **Networking**, perform any of the following operations:
   - To remove a subnet, under **Subnets**, choose **X** for each subnet that you want to remove.
   - To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, select each subnet.
7. To add Amazon EC2 instances to your cluster, expand **Infrastructure**, and then select **Amazon EC2 instances**. Next, configure the Auto Scaling group which acts as the capacity provider:
   - To create a Auto Scaling group, from **Auto Scaling group (ASG)**, select **Create new group**, and then provide the following details about the group:
     - For **Operating system/Architecture**, choose Amazon Linux 2.
     - For **EC2 instance type**, choose `t2.micro`.

       For **SSH key pair**, choose the pair that proves your identity when you connect to the instance.
     - For **Capacity**, enter 1.
8. Choose **Create**.

Step 2: Create a security group for Amazon EC2 instances and the Amazon EFS file system

In this step, you create a security group for your Amazon EC2 instances that allows inbound network traffic on port 80 and your Amazon EFS file system that allows inbound access from your container instances.

Create a security group for your Amazon EC2 instances with the following options:

- **Security group name** - a unique name for your security group.
- **VPC** - the VPC that you identified earlier for your cluster.
- **Inbound rule**
  - **Type** - **HTTP**
  - **Source** - **0.0.0.0/0**.

Create a security group for your Amazon EFS file system with the following options:

- **Security group name** - a unique name for your security group. For example, `EFS-access-for-sg-dc025fa2`.
- **VPC** - the VPC that you identified earlier for your cluster.
- **Inbound rule**
  - **Type** - **NFS**
  - **Source** - **Custom** with the ID of the security group you created for your instances.

For information about how to create a security group, see [Create a security group for your Amazon EC2 instance](#) in the *Amazon EC2 User Guide*.

Step 3: Create an Amazon EFS file system

In this step, you create an Amazon EFS file system.

To create an Amazon EFS file system for Amazon ECS tasks.

1. Open the Amazon Elastic File System console at https://console.aws.amazon.com/efs/.
2. Choose **Create file system**.
3. Enter a name for your file system and then choose the VPC that your container instances are hosted in. By default, each subnet in the specified VPC receives a mount target that uses the default security group for that VPC. Then, choose **Customize**.

> **Note**
>
> This tutorial assumes that your Amazon EFS file system, Amazon ECS cluster, container instances, and tasks are in the same VPC. For more information about mounting a file system from a different VPC, see Walkthrough: Mount a file system from a different VPC in the *Amazon EFS User Guide*.

4. On the **File system settings** page, configure optional settings and then under **Performance settings**, choose the **Bursting** throughput mode for your file system. After you have configured settings, select **Next**.

   a. (Optional) Add tags for your file system. For example, you could specify a unique name for the file system by entering that name in the **Value** column next to the **Name** key.

   b. (Optional) Enable lifecycle management to save money on infrequently accessed storage. For more information, see EFS Lifecycle Management in the *Amazon Elastic File System User Guide*.

   c. (Optional) Enable encryption. Select the check box to enable encryption of your Amazon EFS file system at rest.

5. On the **Network access** page, under **Mount targets**, replace the existing security group configuration for every availability zone with the security group you created for the file system in Step 2: Create a security group for Amazon EC2 instances and the Amazon EFS file system and then choose **Next**.

6. You do not need to configure **File system policy** for this tutorial, so you can skip the section by choosing **Next**.

7. Review your file system options and choose **Create** to complete the process.

8. From the **File systems** screen, record the **File system ID**. In the next step, you will reference this value in your Amazon ECS task definition.

Step 4: Add content to the Amazon EFS file system

In this step, you mount the Amazon EFS file system to an Amazon EC2 instance and add content to it. This is for testing purposes in this tutorial, to illustrate the persistent nature of the data. When using this feature you would normally have your application or another method of writing data to your Amazon EFS file system.

To create an Amazon EC2 instance and mount the Amazon EFS file system

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Launch Instance**.
3. Under **Application and OS Images (Amazon Machine Image)**, select the **Amazon Linux 2 AMI (HVM)**.
4. Under **Instance type**, keep the default instance type, `t2.micro`.
5. Under **Key pair (login)**, select a key pair for SSH access to the instance.
6. Under **Network settings**, select the VPC that you specified for your Amazon EFS file system and Amazon ECS cluster. Select a subnet and the instance security group created in Step 2: Create a security group for Amazon EC2 instances and the Amazon EFS file system. Configure the instance's security group. Ensure that **Auto-assign public IP** is enabled.
7. Under **Configure storage**, choose the **Edit** button for file systems and then choose **EFS**. Select the file system you created in Step 3: Create an Amazon EFS file system. You can optionally change the mount point or leave the default value.

   > **Important**
   >
   > Your must select a subnet before you can add a file system to the instance.

8. Clear the **Automatically create and attach security groups**. Leave the other check box selected. Choose **Add shared file system**.
9. Under **Advanced Details**, ensure that the user data script is populated automatically with the Amazon EFS file system mounting steps.
10. Under **Summary**, ensure the **Number of instances** is **1**. Choose **Launch instance**.
11. On the **Launch an instance** page, choose **View all instances** to see the status of your instances. Initially, the **Instance state** status is `PENDING`. After the state changes to `RUNNING` and the instance passes all status checks, the instance is ready for use.

Now, you connect to the Amazon EC2 instance and add content to the Amazon EFS file system.

To connect to the Amazon EC2 instance and add content to the Amazon EFS file system

1. SSH to the Amazon EC2 instance you created. For more information, see Connect to your Linux instance using SSH in the *Amazon EC2 User Guide*.

2. From the terminal window, run the **df -T** command to verify that the Amazon EFS file system is mounted. In the following output, we have highlighted the Amazon EFS file system mount.

```
$ df -T
Filesystem     Type            1K-blocks    Used         Available Use% Mounted on
devtmpfs       devtmpfs           485468       0            485468   0% /dev
tmpfs          tmpfs              503480       0            503480   0% /dev/shm
tmpfs          tmpfs              503480     424            503056   1% /run
tmpfs          tmpfs              503480       0            503480   0% /sys/fs/
cgroup
/dev/xvda1     xfs               8376300 1310952           7065348  16% /
127.0.0.1:/    nfs4      9007199254739968       0 9007199254739968   0% /mnt/efs/fs1
tmpfs          tmpfs              100700       0            100700   0% /run/
user/1000
```

3. Navigate to the directory that the Amazon EFS file system is mounted at. In the example above, that is `/mnt/efs/fs1`.
4. Create a file named `index.html` with the following content:

```
<html>
    <body>
        <h1>It Works!</h1>
        <p>You are using an Amazon EFS file system for persistent container
 storage.</p>
    </body>
</html>
```

Step 5: Create a task definition

The following task definition creates a data volume named `efs-html`. The `nginx` container mounts the host data volume at the NGINX root, `/usr/share/nginx/html`.

To create a new task definition using the Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, copy and paste the following JSON text, replacing the `fileSystemId` with the ID of your Amazon EFS file system.

```
{
    "containerDefinitions": [
        {
            "memory": 128,
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "mountPoints": [
                {
                    "containerPath": "/usr/share/nginx/html",
                    "sourceVolume": "efs-html"
                }
            ],
            "name": "nginx",
            "image": "public.ecr.aws/docker/library/nginx:latest"
        }
    ],
    "volumes": [
        {
            "name": "efs-html",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1324abcd",
                "transitEncryption": "ENABLED"
            }
        }
    ],
    "family": "efs-tutorial",
    "executionRoleArn":"arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
}
```

Note

The Amazon ECS task execution IAM role does not require any specific Amazon EFS-related permissions to mount an Amazon EFS file system. By default, if no Amazon EFS resource-based policy exists, access is granted to all principals (*) at file system creation.

> The Amazon ECS task role is only required if "EFS IAM authorization" is enabled in the Amazon ECS task definition. When enabled, the task role identity must be allowed access to the Amazon EFS file system in the Amazon EFS resource-based policy, and anonymous access should be disabled.

5. Choose **Create**.

Step 6: Run a task and view the results

Now that your Amazon EFS file system is created and there is web content for the NGINX container to serve, you can run a task using the task definition that you created. The NGINX web server serves your simple HTML page. If you update the content in your Amazon EFS file system, those changes are propagated to any containers that have also mounted that file system.

The task runs in the subnet that you defined for the cluster.

To run a task and view the results using the console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, select the cluster to run the standalone task in.

   Determine the resource from where you launch the service.

To start a service from	Steps	
Clusters	a. On the **Clusters** page, select the cluster to create the service in. b. From the **Tasks** tab, choose **Run new task**.	
Launch type	a. On the **Task** page, choose the task definition. b. If there is more than one revision, select the revision. c. Choose **Create**, **Run task**.	

3. (Optional) Choose how your scheduled task is distributed across your cluster infrastructure. Expand **Compute configuration**, and then do the following:

Distribution method	Steps
Launch type	a. In the **Compute options** section, select **Launch type**. b. For **Launch type**, choose **EC2**.

4. For **Application type**, choose **Task**.
5. For **Task definition**, choose the `efs-tutorial` task definition that you created earlier.
6. For **Desired tasks**, enter 1.
7. Choose **Create**.
8. On the **Cluster** page, choose **Infrastructure**.
9. Under **Container Instances**, choose the container instance to connect to.
10. On the **Container Instance** page, under **Networking**, record the **Public IP** for your instance.
11. Open a browser and enter the public IP address. You should see the following message:

```
It works!
You are using an Amazon EFS file system for persistent container storage.
```

> **Note**
>
> If you do not see the message, make sure that the security group for your container instance allows inbound network traffic on port 80 and the security group for your file system allows inbound access from the container instance.

Use FSx for Windows File Server volumes with Amazon ECS

FSx for Windows File Server provides fully managed Windows file servers, that are backed by a Windows file system. When using FSx for Windows File Server together with ECS, you can provision your Windows tasks with persistent, distributed, shared, static file storage. For more information, see [What Is FSx for Windows File Server?](#).

> **Note**
>
> EC2 instances that use the Amazon ECS-Optimized Windows Server 2016 Full AMI do not support FSx for Windows File Server ECS task volumes.
> You can't use FSx for Windows File Server volumes in a Windows containers on Fargate configuration. Instead, you can modify containers to mount them on startup.

You can use FSx for Windows File Server to deploy Windows workloads that require access to shared external storage, highly-available Regional storage, or high-throughput storage. You can mount one or more FSx for Windows File Server file system volumes to an Amazon ECS container that runs on an Amazon ECS Windows instance. You can share FSx for Windows File Server file system volumes between multiple Amazon ECS containers within a single Amazon ECS task.

To enable the use of FSx for Windows File Server with ECS, include the FSx for Windows File Server file system ID and the related information in a task definition. This is in the following example task definition JSON snippet. Before you create and run a task definition, you need the following.

- An ECS Windows EC2 instance that's joined to a valid domain. It can be hosted by an AWS Directory Service for Microsoft Active Directory, on-premises Active Directory or self-hosted Active Directory on Amazon EC2.
- An AWS Secrets Manager secret or Systems Manager parameter that contains the credentials that are used to join the Active Directory domain and attach the FSx for Windows File Server file system. The credential values are the name and password credentials that you entered when creating the Active Directory.

For a related tutorial, see Learn how to configure FSx for Windows File Server file systems for Amazon ECS.

Considerations

Consider the following when using FSx for Windows File Server volumes:

- FSx for Windows File Server with Amazon ECS only supports Windows Amazon EC2 instances. Linux Amazon EC2 instances aren't supported.
- FSx for Windows File Server with Amazon ECS doesn't support AWS Fargate.
- FSx for Windows File Server with Amazon ECS with `awsvpc` network mode requires version `1.54.0` or later of the container agent.

- The maximum number of drive letters that can be used for an Amazon ECS task is 23. Each task with an FSx for Windows File Server volume gets a drive letter assigned to it.
- By default, task resource cleanup time is three hours after the task ended. Even if no tasks use it, a file mapping that's created by a task persists for three hours. The default cleanup time can be configured by using the Amazon ECS environment variable `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`. For more information, see Amazon ECS container agent configuration.
- Tasks typically only run in the same VPC as the FSx for Windows File Server file system. However, it's possible to have cross-VPC support if there's an established network connectivity between the Amazon ECS cluster VPC and the FSx for Windows File Server file-system through VPC peering.
- You control access to an FSx for Windows File Server file system at the network level by configuring the VPC security groups. Only tasks that are hosted on EC2 instances joined to the Active Directory domain with correctly configured Active Directory security groups can access the FSx for Windows File Server file-share. If the security groups are misconfigured, Amazon ECS fails to launch the task with the following error message: `unable to mount file system` *`fs-id`*."
- FSx for Windows File Server is integrated with AWS Identity and Access Management (IAM) to control the actions that your IAM users and groups can take on specific FSx for Windows File Server resources. With client authorization, customers can define IAM roles that allow or deny access to specific FSx for Windows File Server file systems, optionally require read-only access, and optionally allow or disallow root access to the file system from the client. For more information, see Security in the Amazon FSx Windows User Guide.

Best practices for using FSx for Windows File Server with Amazon ECS

Make note of the following best practice recommendations when you use FSx for Windows File Server with Amazon ECS.

Security and access controls for FSx for Windows File Server

FSx for Windows File Server offers the following access control features that you can use to ensure that the data stored in an FSx for Windows File Server file system is secure and accessible only from applications that need it.

Data encryption for FSx for Windows File Server volumes

FSx for Windows File Server supports two forms of encryption for file systems. They are encryption of data in transit and encryption at rest. Encryption of data in transit is supported on file shares that are mapped on a container instance that supports SMB protocol 3.0 or newer. Encryption of data at rest is automatically enabled when creating an Amazon FSx file system. Amazon FSx automatically encrypts data in transit using SMB encryption as you access your file system without the need for you to modify your applications. For more information, see [Data encryption in Amazon FSx](#) in the *Amazon FSx for Windows File Server User Guide*.

Use Windows ACLs for folder level access control

The Windows Amazon EC2 instance access Amazon FSx file shares using Active Directory credentials. It uses standard Windows access control lists (ACLs) for fine-grained file-level and folder-level access control. You can create multiple credentials, each one for a specific folder within the share which maps to a specific task.

In the following example, the task has access to the folder `App01` using a credential saved in Secrets Manager. Its Amazon Resource Name (ARN) is `1234`.

```
"rootDirectory": "\\path\\to\\my\\data\App01",
"credentialsParameter": "arn-1234",
"domain": "corp.fullyqualified.com",
```

In another example, a task has access to the folder `App02` using a credential saved in the Secrets Manager. Its ARN is `6789`.

```
"rootDirectory": "\\path\\to\\my\\data\App02",
"credentialsParameter": "arn-6789",
"domain": "corp.fullyqualified.com",
```

Specify an FSx for Windows File Server file system in an Amazon ECS task definition

To use FSx for Windows File Server file system volumes for your containers, specify the volume and mount point configurations in your task definition. The following task definition JSON snippet shows the syntax for the `volumes` and `mountPoints` objects for a container.

```
{
    "containerDefinitions": [
        {
            "entryPoint": [
```

```
                "powershell",
                "-Command"
            ],
            "portMappings": [],
            "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>It Works!</h2> <p>You are using Amazon
 FSx for Windows File Server file system for persistent container storage.</p>' -
Force"],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": false,
            "name": "container1",
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ]
        },
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [
                {
                    "hostPort": 443,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": ["Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force; Start-
Sleep -Seconds 120; Move-Item -Path C:\\fsx-windows-dir\\index.html -Destination C:\
\inetpub\\wwwroot\\index.html -Force; C:\\ServiceMonitor.exe w3svc"],
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
```

```
                }
            ],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": true,
            "name": "container2"
        }
    ],
    "family": "fsx-windows",
    "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
    "volumes": [
        {
            "name": "fsx-windows-dir",
            "fsxWindowsFileServerVolumeConfiguration": {
                "fileSystemId": "fs-0eeb5730b2EXAMPLE",
                "authorizationConfig": {
                    "domain": "example.com",
                    "credentialsParameter": "arn:arn-1234"
                },
                "rootDirectory": "share"
            }
        }
    ]
}
```

`FSxWindowsFileServerVolumeConfiguration`

Type: Object

Required: No

This parameter is specified when you're using FSx for Windows File Server file system for task storage.

`fileSystemId`

Type: String

Required: Yes

The FSx for Windows File Server file system ID to use.

`rootDirectory`

Type: String

Required: Yes

The directory within the FSx for Windows File Server file system to mount as the root directory inside the host.

`authorizationConfig`

`credentialsParameter`

Type: String

Required: Yes

The authorization credential options:

- Amazon Resource Name (ARN) of an Secrets Manager secret.
- Amazon Resource Name (ARN) of an Systems Manager parameter.

`domain`

Type: String

Required: Yes

A fully qualified domain name that's hosted by an AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) directory or a self-hosted EC2 Active Directory.

Methods for storing FSx for Windows File Server volume credentials

There are two different methods of storing credentials for use with the credentials parameter.

- **AWS Secrets Manager secret**

  This credential can be created in the AWS Secrets Manager console by using the *Other type of secret* category. You add a row for each key/value pair, username/admin and password/*`password`*.
- **Systems Manager parameter**

This credential can be created in the Systems Manager parameter console by entering text in the form that's in the following example code snippet.

```
{
  "username": "admin",
  "password": "password"
}
```

The `credentialsParameter` in the task definition `FSxWindowsFileServerVolumeConfiguration` parameter holds either the secret ARN or the Systems Manager parameter ARN. For more information, see What is AWS Secrets Manager in the *Secrets Manager User Guide* and Systems Manager Parameter Store from the *Systems Manager User Guide*.

Learn how to configure FSx for Windows File Server file systems for Amazon ECS

Learn how to launch an Amazon ECS-Optimized Windows instance that hosts an FSx for Windows File Server file system and containers that can access the file system. To do this, you first create an AWS Directory Service AWS Managed Microsoft Active Directory. Then, you create an FSx for Windows File Server File Server file system and cluster with an Amazon EC2 instance and a task definition. You configure the task definition for your containers to use the FSx for Windows File Server file system. Finally, you test the file system.

It takes 20 to 45 minutes each time you launch or delete either the Active Directory or the FSx for Windows File Server file system. Be prepared to reserve at least 90 minutes to complete the tutorial or complete the tutorial over a few sessions.

Prerequisites for the tutorial

- An administrative user. See Set up to use Amazon ECS.
- (Optional) A PEM key pair for connecting to your EC2 Windows instance through RDP access. For information about how to create key pairs, see Amazon EC2 key pairs and Amazon EC2 instances in the *Amazon EC2 User Guide*.
- A VPC with at least one public and one private subnet, and one security group. You can use your default VPC. You don't need a NAT gateway or device. AWS Directory Service doesn't support Network Address Translation (NAT) with Active Directory. For this to work, the Active Directory, FSx for Windows File Server file system, ECS Cluster, and EC2 instance must be located within

your VPC. For more information regarding VPCs and Active Directories, see Create a VPC and Prerequisites for creating an AWS Managed Microsoft AD.

- The IAM ecsInstanceRole and ecsTaskExecutionRole permissions are associated with your account. These service-linked roles allow services to make API calls and access containers, secrets, directories, and file servers on your behalf.

Step 1: Create IAM access roles

Create a cluster with the AWS Management Console.

1. See Amazon ECS container instance IAM role to check whether you have an ecsInstanceRole and to see how you can create one if you don't have one.
2. We recommend that role policies are customized for minimum permissions in an actual production environment. For the purpose of working through this tutorial, verify that the following AWS managed policy is attached to your ecsInstanceRole. Attach the policy if it is not already attached.

   - AmazonEC2ContainerServiceforEC2Role
   - AmazonSSMManagedInstanceCore
   - AmazonSSMDirectoryServiceAccess

   To attach AWS managed policies.

   a. Open the IAM console.
   b. In the navigation pane, choose **Roles.**
   c. Choose an **AWS managed role**.
   d. Choose **Permissions, Attach policies**.
   e. To narrow the available policies to attach, use **Filter**.
   f. Select the appropriate policy and choose **Attach policy**.
3. See Amazon ECS task execution IAM role to check whether you have an ecsTaskExecutionRole and to see how you can create one if you don't have one.

   We recommend that role policies are customized for minimum permissions in an actual production environment. For the purpose of working through this tutorial, verify that the following AWS managed policies are attached to your ecsTaskExecutionRole. Attach the policies if they are not already attached. Use the procedure given in the preceding section to attach the AWS managed policies.

- SecretsManagerReadWrite
- AmazonFSxReadOnlyAccess
- AmazonSSMReadOnlyAccess
- AmazonECSTaskExecutionRolePolicy

Step 2: Create Windows Active Directory (AD)

1. Follow the steps described in Creating your AWS Managed Microsoft AD in the AWS *Directory Service Administration Guide*. Use the VPC you have designated for this tutorial. On Step 3 of *Creating your AWS Managed Microsoft AD*, save the user name and admin password for use in a following step. Also, note the fully qualified directory DNS name for future steps. You can complete the following step while the Active Directory is being created.
2. Create an AWS Secrets Manager secret to use in the following steps. For more information, see Get started with Secrets Manager in the AWS *Secrets Manager User Guide*.

   a. Open the Secrets Manager console.

   b. Click **Store a new secret**.

   c. Select **Other type of secrets**.

   d. For **Secret key/value**, in the first row, create a key **`username`** with value **`admin`**. Click on **+ Add row**.

   e. In the new row, create a key **`password`**. For value, type in the password you entered in Step 3 of *Create Your AWS Managed AD Directory*.

   f. Click on the **Next** button.

   g. Provide a secret name and description. Click **Next**.

   h. Click **Next**. Click **Store**.

   i. From the list of **Secrets** page, click on the secret you have just created.

   j. Save the ARN of the new secret for use in the following steps.

   k. You can proceed to the next step while your Active Directory is being created.

Step 3: Verify and update your security group

In this step, you verify and update the rules for the security group that you're using. For this, you can use the default security group that was created for your VPC.

Verify and update security group.

You need to create or edit your security group to send data from and to the ports, which are described in [Amazon VPC Security Groups] in the *FSx for Windows File Server User Guide*. You can do this by creating the security group inbound rule shown in the first row of the following table of inbound rules. This rule allows inbound traffic from network interfaces (and their associated instances) that are assigned to the security group. All of the cloud resources you create are within the same VPC and attached to the same security group. Therefore, this rule allows traffic to be sent to and from the FSx for Windows File Server file system, Active Directory, and ECS instance as required. The other inbound rules allow traffic to serve the website and RDP access for connecting to your ECS instance.

The following table shows which security group inbound rules are required for this tutorial.

Type	Protocol	Port range	Source
All traffic	All	All	*`sg-securitygroup`*
HTTPS	TCP	443	0.0.0.0/0
RDP	TCP	3389	your laptop IP address

The following table shows which security group outbound rules are required for this tutorial.

Type	Protocol	Port range	Destination
All traffic	All	All	0.0.0.0/0

1. Open the EC2 console and select **Security Groups** from the left-hand menu.
2. From the list of security groups now displayed, select check the check-box to the left of the security group that you are using for this tutorial.

   Your security group details are displayed.
3. Edit the inbound and outbound rules by selecting the **Inbound rules** or **Outbound rules** tabs and choosing the **Edit inbound rules** or **Edit outbound rules** buttons. Edit the rules to match those displayed in the preceding tables. After you create your EC2 instance later on in this tutorial, edit the inbound rule RDP source with the public IP address of your EC2 instance as described in Connect to your Windows instance using RDP from the *Amazon EC2 User Guide*.

Step 4: Create an FSx for Windows File Server file system

After your security group is verified and updated and your Active Directory is created and is in the active status, create the FSx for Windows File Server file system in the same VPC as your Active Directory. Use the following steps to create an FSx for Windows File Server file system for your Windows tasks.

Create your first file system.

1. Open the Amazon FSx console.
2. On the dashboard, choose **Create file system** to start the file system creation wizard.
3. On the **Select file system type** page, choose **FSx for Windows File Server**, and then choose **Next**. The **Create file system** page appears.
4. In the **File system details** section, provide a name for your file system. Naming your file systems makes it easier to find and manage your them. You can use up to 256 Unicode characters. Allowed characters are letters, numbers, spaces, and the special characters plus sign (+). minus sign (-), equal sign (=), period (.), underscore (_), colon (:), and forward slash (/).
5. For **Deployment type** choose **Single-AZ** to deploy a file system that is deployed in a single Availability Zone. *Single-AZ 2* is the latest generation of single Availability Zone file systems, and it supports SSD and HDD storage.
6. For **Storage type**, choose **HDD**.
7. For **Storage capacity**, enter the minimum storage capacity.
8. Keep **Throughput capacity** at its default setting.
9. In the **Network & security** section, choose the same Amazon VPC that you chose for your AWS Directory Service directory.

10. For **VPC Security Groups**, choose the security group that you verified in *Step 3: Verify and update your security group*.
11. For **Windows authentication**, choose **AWS Managed Microsoft Active Directory**, and then choose your AWS Directory Service directory from the list.
12. For **Encryption**, keep the default **Encryption key** setting of **aws/fsx (default)**.
13. Keep the default settings for **Maintenance preferences**.
14. Click on the **Next** button.
15. Review the file system configuration shown on the **Create file system** page. For your reference, note which file system settings you can modify after file system is created. Choose **Create file system**.
16. Note the file system ID. You will need to use it in a later step.

    You can go on to the next steps to create a cluster and EC2 instance while the FSx for Windows File Server file system is being created.

Step 5: Create an Amazon ECS cluster

Create a cluster using the Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, for **Cluster name**, enter **windows-fsx-cluster**.
6. Expand **Infrastructure**, clear AWS Fargate (serverless) and then select **Amazon EC2 instances**.

    - To create a Auto Scaling group, from **Auto Scaling group (ASG)**, select **Create new group**, and then provide the following details about the group:

        - For **Operating system/Architecture**, choose **Windows Server 2019 Core**.
        - For **EC2 instance type**, choose t2.medium or t2.micro.
7. Choose **Create**.

Step 6: Create an Amazon ECS optimized Amazon EC2 instance

Create an Amazon ECS Windows container instance.

To create an Amazon ECS instance

1. Use the `aws ssm get-parameters` command to retrieve the AMI name for the Region that hosts your VPC. For more information, see Retrieving Amazon ECS-Optimized AMI metadata.
2. Use the Amazon EC2 console to launch the instance.

    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

    b. From the navigation bar, select the Region to use.

    c. From the **EC2 Dashboard**, choose **Launch instance**.

    d. For **Name**, enter a unique name.

    e. For **Application and OS Images (Amazon Machine Image)**, in the **search** field, enter the AMI name that you retrieved.

    f. For **Instance type**, choose t2.medium or t2.micro.

    g. For **Key pair (login)**, choose a key pair. If you don't specify a key pair, you

    h. Under **Network settings**, for **VPC** and **Subnet**, choose your VPC and a public subnet.

    i. Under **Network settings**, for **Security group**, choose an existing security group, or create a new one. Ensure that the security group you choose has the inbound and outbound rules defined in Prerequisites for the tutorial

    j. Under **Network settings**, for **Auto-assign Public IP**, select **Enable**.

    k. Expand **Advanced details**, and then for **Domain join directory**, select the ID of the Active Directory that you created. This option domain joins your AD when the EC2 instance is launched.

    l. Under **Advanced details**, for **IAM instance profile** , choose **ecsInstanceRole**.

    m. Configure your Amazon ECS container instance with the following user data. Under **Advanced Details**, paste the following script into the **User data** field, replacing *`cluster_name`* with the name of your cluster.

    ```
    <powershell>
    Initialize-ECSAgent -Cluster windows-fsx-cluster -EnableTaskIAMRole
    </powershell>
    ```

    n. When you are ready, select the acknowledgment field, and then choose **Launch Instances**.

o. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.

3. Open the console at https://console.aws.amazon.com/ecs/v2.
4. In the navigation pane, choose **Clusters**, and then choose **windows-fsx-cluster**.
5. Choose the **Infrastructure** tab and verify that your instance has been registered in the **windows-fsx-cluster** cluster.

Step 7: Register a Windows task definition

Before you can run Windows containers in your Amazon ECS cluster, you must register a task definition. The following task definition example displays a simple web page. The task launches two containers that have access to the FSx file system. The first container writes an HTML file to the file system. The second container downloads the HTML file from the file system and serves the webpage.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, replace the values for your task execution role and the details about your FSx file system and then choose **Save**.

```
{
    "containerDefinitions": [
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [],
            "command": ["New-Item -Path C:\\fsx-windows-dir\\index.html -ItemType
 file -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body
 {margin-top: 40px; background-color: #333;} </style> </head><body> <div
 style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>It
 Works!</h2> <p>You are using Amazon FSx for Windows File Server file system for
 persistent container storage.</p>' -Force"],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
```

```
            "essential": false,
            "name": "container1",
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ]
        },
        {
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "portMappings": [
                {
                    "hostPort": 443,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": ["Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force;
 Start-Sleep -Seconds 120; Move-Item -Path C:\\fsx-windows-dir\\index.html -
Destination C:\\inetpub\\wwwroot\\index.html -Force; C:\\ServiceMonitor.exe
 w3svc"],
            "mountPoints": [
                {
                    "sourceVolume": "fsx-windows-dir",
                    "containerPath": "C:\\fsx-windows-dir",
                    "readOnly": false
                }
            ],
            "cpu": 512,
            "memory": 256,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "essential": true,
            "name": "container2"
        }
    ],
    "family": "fsx-windows",
    "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
    "volumes": [
```

```
            {
                "name": "fsx-windows-dir",
                "fsxWindowsFileServerVolumeConfiguration": {
                    "fileSystemId": "fs-0eeb5730b2EXAMPLE",
                    "authorizationConfig": {
                        "domain": "example.com",
                        "credentialsParameter": "arn:arn-1234"
                    },
                    "rootDirectory": "share"
                }
            }
        ]
}
```

Step 8: Run a task and view the results

Before running the task, verify that the status of your FSx for Windows File Server file system is **Available**. After it is available, you can run a task using the task definition that you created. The task starts out by creating containers that shuffle an HTML file between them using the file system. After the shuffle, a web server serves the simple HTML page.

> **Note**
>
> You might not be able to connect to the website from within a VPN.

Run a task and view the results with the Amazon ECS console.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**, and then choose **windows-fsx-cluster**.
3. Choose the **Tasks** tab, and then choose **Run new task**.
4. For **Launch Type**, choose **EC2**.
5. Under Deployment configuration, for **Task Definition**, choose the **fsx-windows**, and then choose **Create**.
6. When your task status is **RUNNING**, choose the task ID.
7. Under **Containers**, when the container1 status is **STOPPED**, select container2 to view the container's details.

8. Under **Container details for container2**, select **Network bindings** and then click on the external IP address that is associated with the container. Your browser will open and display the following message.

```
Amazon ECS Sample App
It Works!
You are using Amazon FSx for Windows File Server file system for persistent
 container storage.
```

> **Note**
>
> It may take a few minutes for the message to be displayed. If you don't see this message after a few minutes, check that you aren't running in a VPN and make sure that the security group for your container instance allows inbound network HTTP traffic on port 443.

Step 9: Clean up

> **Note**
>
> It takes 20 to 45 minutes to delete the FSx for Windows File Server file system or the AD. You must wait until the FSx for Windows File Server file system delete operations are complete before starting the AD delete operations.

Delete FSx for Windows File Server file system.

1. Open the [Amazon FSx console]
2. Choose the radio button to the left of the FSx for Windows File Server file system that you just created.
3. Choose **Actions**.
4. Select **Delete file system**.

Delete AD.

1. Open the [AWS Directory Service console].

2. Choose the radio button to the left of the AD you just created.
3. Choose **Actions**.
4. Select **Delete directory**.

Delete the cluster.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**, and then choose **windows-fsx-cluster**.
3. Choose **Delete cluster**.
4. Enter the phrase and then choose **Delete**.

Terminate EC2 instance.

1. Open the Amazon EC2 console.
2. From the left-hand menu, select **Instances**.
3. Check the box to the left of the EC2 instance you created.
4. Click the **Instance state**, **Terminate instance**.

Delete secret.

1. Open the Secrets Manager console.
2. Select the secret you created for this walk through.
3. Click **Actions**.
4. Select **Delete secret**.

Use Docker volumes with Amazon ECS

When using Docker volumes, the built-in `local` driver or a third-party volume driver can be used. Docker volumes are managed by Docker and a directory is created in `/var/lib/docker/volumes` on the container instance that contains the volume data.

To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see Volumes in the Docker documentation.

Some common use cases for Docker volumes are the following:

- To provide persistent data volumes for use with containers
- To share a defined data volume at different locations on different containers on the same container instance
- To define an empty, nonpersistent data volume and mount it on multiple containers within the same task
- To provide a data volume to your task that's managed by a third-party driver

Considerations for using Docker volumes

Consider the following when using Docker volumes:

- Docker volumes are only supported when using the EC2 launch type or external instances.
- Windows containers only support the use of the `local` driver.
- If a third-party driver is used, make sure it's installed and active on the container instance before the container agent is started. If the third-party driver isn't active before the agent is started, you can restart the container agent using one of the following commands:
  - For the Amazon ECS-optimized Amazon Linux 2 AMI:

    ```
    sudo systemctl restart ecs
    ```
  - For the Amazon ECS-optimized Amazon Linux AMI:

    ```
    sudo stop ecs && sudo start ecs
    ```

For information about how to specify a Docker volume in a task definition, see Specify a Docker volume in an Amazon ECS task definition.

Specify a Docker volume in an Amazon ECS task definition

Before your containers can use data volumes, you must specify the volume and mount point configurations in your task definition. This section describes the volume configuration for a container. For tasks that use a Docker volume, specify a `dockerVolumeConfiguration`. For tasks that use a bind mount host volume, specify a `host` and optional `sourcePath`.

The following task definition JSON shows the syntax for the `volumes` and `mountPoints` objects for a container.

```
{
    "containerDefinitions": [
        {
            "mountPoints": [
                {
                    "sourceVolume": "string",
                    "containerPath": "/path/to/mount_volume",
                    "readOnly": boolean
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "string",
            "dockerVolumeConfiguration": {
                "scope": "string",
                "autoprovision": boolean,
                "driver": "string",
                "driverOpts": {
                    "key": "value"
                },
                "labels": {
                    "key": "value"
                }
            }
        }
    ]
}
```

`name`

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens (-), and underscores (_) are allowed. This name is referenced in the `sourceVolume` parameter of the container definition `mountPoints` object.

`dockerVolumeConfiguration`

Type: DockerVolumeConfiguration Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are supported only when running tasks on EC2 instances. Windows containers support only the use of the `local` driver. To use bind mounts, specify a `host` instead.

`scope`

Type: String

Valid Values: `task` | `shared`

Required: No

The scope for the Docker volume, which determines its lifecycle. Docker volumes that are scoped to a `task` are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as `shared` persist after the task stops.

`autoprovision`

Type: Boolean

Default value: `false`

Required: No

If this value is `true`, the Docker volume is created if it doesn't already exist. This field is used only if the `scope` is `shared`. If the `scope` is `task`, then this parameter must be omitted.

`driver`

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because this name is used for task placement. If the driver was installed by using the Docker plugin CLI, use `docker plugin ls` to retrieve the driver name from your container instance. If the driver was installed by using another method, use Docker plugin discovery to retrieve the driver name.

`driverOpts`

Type: String

Required: No

A map of Docker driver-specific options to pass through. This parameter maps to `DriverOpts` in the Create a volume section of Docker.

`labels`

Type: String

Required: No

Custom metadata to add to your Docker volume.

`mountPoints`

Type: Object array

Required: No

The mount points for the data volumes in your container. This parameter maps to `Volumes` in the create-container Docker API and the `--volume` option to docker run.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount points cannot be used across drives. You must specify mount points to attach an Amazon EBS volume directly to an Amazon ECS task.

`sourceVolume`

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

`containerPath`

Type: String

Required: Yes, when `mountPoints` are used

The path in the container where the volume will be mounted.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

For tasks that run on EC2 instances running the Windows operating system, leave the value as the default of `false`.

Docker volume examples for Amazon ECS

The following examples show how to provide ephemeral storage for a container and how to provide a shared volume for multiple conatiners, and how to provide NFS persistent storage for a container.

To provide ephemeral storage for a container using a Docker volume

In this example, a container uses an empty data volume that is disposed of after the task is finished. One example use case is that you might have a container that needs to access some scratch file storage location during a task. This task can be achieved using a Docker volume.

1. In the task definition `volumes` section, define a data volume with `name` and `DockerVolumeConfiguration` values. In this example, we specify the scope as `task` so the volume is deleted after the task stops and use the built-in `local` driver.

```
"volumes": [
    {
        "name": "scratch",
        "dockerVolumeConfiguration" : {
            "scope": "task",
            "driver": "local",
            "labels": {
                "scratch": "space"
            }
        }
    }
]
```

2. In the `containerDefinitions` section, define a container with `mountPoints` values that reference the name of the defined volume and the `containerPath` value to mount the volume at on the container.

```
"containerDefinitions": [
    {
```

```
        "name": "container-1",
        "mountPoints": [
            {
              "sourceVolume": "scratch",
              "containerPath": "/var/scratch"
            }
        ]
    }
]
```

To provide persistent storage for multiple containers using a Docker volume

In this example, you want a shared volume for multiple containers to use and you want it to persist after any single task that use it stopped. The built-in `local` driver is being used. This is so the volume is still tied to the lifecycle of the container instance.

1. In the task definition `volumes` section, define a data volume with `name` and `DockerVolumeConfiguration` values. In this example, specify a `shared` scope so the volume persists, set autoprovision to `true`. This is so that the volume is created for use. Then, also use the built-in `local` driver.

```
"volumes": [
    {
        "name": "database",
        "dockerVolumeConfiguration" : {
            "scope": "shared",
            "autoprovision": true,
            "driver": "local",
            "labels": {
                "database": "database_name"
            }
        }
    }
]
```

2. In the `containerDefinitions` section, define a container with `mountPoints` values that reference the name of the defined volume and the `containerPath` value to mount the volume at on the container.

```
"containerDefinitions": [
    {
```

```
        "name": "container-1",
        "mountPoints": [
        {
          "sourceVolume": "database",
          "containerPath": "/var/database"
        }
      ]
    },
    {
      "name": "container-2",
      "mountPoints": [
        {
          "sourceVolume": "database",
          "containerPath": "/var/database"
        }
      ]
    }
  ]
```

To provide NFS persistent storage for a container using a Docker volume

In this example, a container uses an NFS data volume that is automatically mounted when the task starts and unmounted when the task stops. This uses the Docker built-in `local` driver. One example use case is that you might have a local NFS storage and need to access it from an ECS Anywhere task. This can be achieved using a Docker volume with NFS driver option.

1. In the task definition `volumes` section, define a data volume with `name` and `DockerVolumeConfiguration` values. In this example, specify a `task` scope so the volume is unmounted after the task stops. Use the `local` driver and configure the `driverOpts` with the `type`, `device`, and o options accordingly. Replace NFS_SERVER with the NFS server endpoint.

```
"volumes": [
       {
           "name": "NFS",
           "dockerVolumeConfiguration" : {
               "scope": "task",
               "driver": "local",
               "driverOpts": {
                   "type": "nfs",
                   "device": "$NFS_SERVER:/mnt/nfs",
```

```
                    "o": "addr=$NFS_SERVER"
                }
            }
        }
    ]
```

2. In the `containerDefinitions` section, define a container with `mountPoints` values that reference the name of the defined volume and the `containerPath` value to mount the volume on the container.

```
"containerDefinitions": [
        {
            "name": "container-1",
            "mountPoints": [
                {
                  "sourceVolume": "NFS",
                  "containerPath": "/var/nfsmount"
                }
            ]
        }
    ]
```

Use bind mounts with Amazon ECS

With bind mounts, a file or directory on a host, such as an Amazon EC2 instance, is mounted into a container. Bind mounts are supported for tasks that are hosted on both Fargate and Amazon EC2 instances. Bind mounts are tied to the lifecycle of the container that uses them. After all of the containers that use a bind mount are stopped, such as when a task is stopped, the data is removed. For tasks that are hosted on Amazon EC2 instances, the data can be tied to the lifecycle of the host Amazon EC2 instance by specifying a `host` and optional `sourcePath` value in your task definition. For more information, see [Bind mounts](#) in the Docker documentation.

The following are common use cases for bind mounts.

- To provide an empty data volume to mount in one or more containers.
- To mount a host data volume in one or more containers.
- To share a data volume from a source container with other containers in the same task.
- To expose a path and its contents from a Dockerfile to one or more containers.

Considerations when using bind mounts

When using bind mounts, consider the following.

- By default, tasks that are hosted on AWS Fargate using platform version `1.4.0` or later (Linux) or `1.0.0` or later (Windows) receive a minimum of 20 GiB of ephemeral storage for bind mounts. You can increase the total amount of ephemeral storage up to a maximum of 200 GiB by specifying the `ephemeralStorage` parameter in your task definition.
- To expose files from a Dockerfile to a data volume when a task is run, the Amazon ECS data plane looks for a `VOLUME` directive. If the absolute path that's specified in the `VOLUME` directive is the same as the `containerPath` that's specified in the task definition, the data in the `VOLUME` directive path is copied to the data volume. In the following Dockerfile example, a file that's named `examplefile` in the `/var/log/exported` directory is written to the host and then mounted inside the container.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN mkdir -p /var/log/exported
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

  By default, the volume permissions are set to `0755` and the owner as `root`. You can customize these permissions in the Dockerfile. The following example defines the owner of the directory as node.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
RUN touch /var/log/exported/examplefile
USER node
VOLUME ["/var/log/exported"]
```

- For tasks that are hosted on Amazon EC2 instances, when a `host` and `sourcePath` value aren't specified, the Docker daemon manages the bind mount for you. When no containers reference this bind mount, the Amazon ECS container agent task cleanup service eventually deletes it. By default, this happens three hours after the container exits. However, you can configure this duration with the `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` agent variable. For more information, see [Amazon ECS container agent configuration](). If you need this data to persist beyond the lifecycle of the container, specify a `sourcePath` value for the bind mount.

Specify a bind mount in an Amazon ECS task definition

For Amazon ECS tasks that are hosted on either Fargate or Amazon EC2 instances, the following task definition JSON snippet shows the syntax for the `volumes`, `mountPoints`, and `ephemeralStorage` objects for a task definition.

```
{
   "family": "",
   ...
   "containerDefinitions" : [
      {
         "mountPoints" : [
            {
               "containerPath" : "/path/to/mount_volume",
               "sourceVolume" : "string"
            }
          ],
          "name" : "string"
       }
    ],
    ...
    "volumes" : [
       {
          "name" : "string"
       }
    ],
    "ephemeralStorage": {
    "sizeInGiB": integer
    }
}
```

For Amazon ECS tasks that are hosted on Amazon EC2 instances, you can use the optional `host` parameter and a `sourcePath` when specifying the task volume details. When it's specified, it ties the bind mount to the lifecycle of the task rather than the container.

```
"volumes" : [
    {
        "host" : {
            "sourcePath" : "string"
        },
        "name" : "string"
    }
```

```
]
```

The following describes each task definition parameter in more detail.

`name`

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens (-), and underscores (_) are allowed. This name is referenced in the `sourceVolume` parameter of the container definition `mountPoints` object.

`host`

Required: No

The `host` parameter is used to tie the lifecycle of the bind mount to the host Amazon EC2 instance, rather than the task, and where it is stored. If the `host` parameter is empty, then the Docker daemon assigns a host path for your data volume, but the data is not guaranteed to persist after the containers associated with it stop running.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`.

> **Note**
>
> The `sourcePath` parameter is supported only when using tasks that are hosted on Amazon EC2 instances.

`sourcePath`

Type: String

Required: No

When the `host` parameter is used, specify a `sourcePath` to declare the path on the host Amazon EC2 instance that is presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you. If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host Amazon EC2 instance until you delete it manually. If the `sourcePath` value does not exist on the host Amazon EC2 instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

`mountPoints`

Type: Object array

Required: No

The mount points for the data volumes in your container. This parameter maps to `Volumes` in the create-container Docker API and the `--volume` option to docker run.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount points cannot be used across drives. You must specify mount points to attach an Amazon EBS volume directly to an Amazon ECS task.

`sourceVolume`

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

`containerPath`

Type: String

Required: Yes, when `mountPoints` are used

The path in the container where the volume will be mounted.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

For tasks that run on EC2 instances running the Windows operating system, leave the value as the default of `false`.

`ephemeralStorage`

Type: Object

Required: No

The amount of ephemeral storage to allocate for the task. This parameter is used to expand the total amount of ephemeral storage available, beyond the default amount, for tasks hosted on AWS Fargate using platform version `1.4.0` or later (Linux) or `1.0.0` or later (Windows).

You can use the Copilot CLI, CloudFormation, the AWS SDK or the CLI to specify ephemeral storage for a bind mount.

Bind mount examples for Amazon ECS

The following examples cover the common use cases for using a bind mount for your containers.

To allocate an increased amount of ephemeral storage space for a Fargate task

For Amazon ECS tasks that are hosted on Fargate using platform version `1.4.0` or later (Linux) or `1.0.0` (Windows), you can allocate more than the default amount of ephemeral storage for the containers in your task to use. This example can be incorporated into the other examples to allocate more ephemeral storage for your Fargate tasks.

- In the task definition, define an `ephemeralStorage` object. The `sizeInGiB` must be an integer between the values of `21` and `200` and is expressed in GiB.

```
"ephemeralStorage": {
    "sizeInGiB": integer
}
```

To provide an empty data volume for one or more containers

In some cases, you want to provide the containers in a task some scratch space. For example, you might have two database containers that need to access the same scratch file storage location during a task. This can be achieved using a bind mount.

1. In the task definition `volumes` section, define a bind mount with the name `database_scratch`.

```
"volumes": [
  {
    "name": "database_scratch"
  }
]
```

2. In the `containerDefinitions` section, create the database container definitions. This is so that they mount the volume.

```
"containerDefinitions": [
    {
      "name": "database1",
      "image": "my-repo/database",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "database_scratch",
          "containerPath": "/var/scratch"
        }
      ]
    },
    {
      "name": "database2",
      "image": "my-repo/database",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "database_scratch",
          "containerPath": "/var/scratch"
        }
      ]
    }
  ]
```

To expose a path and its contents in a Dockerfile to a container

In this example, you have a Dockerfile that writes data that you want to mount inside a container. This example works for tasks that are hosted on Fargate or Amazon EC2 instances.

1. Create a Dockerfile. The following example uses the public Amazon Linux 2 container image and creates a file that's named `examplefile` in the `/var/log/exported` directory that we want to mount inside the container. The `VOLUME` directive should specify an absolute path.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN mkdir -p /var/log/exported
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

   By default, the volume permissions are set to `0755` and the owner as `root`. These permissions can be changed in the Dockerfile. In the following example, the owner of the `/var/log/exported` directory is set to node.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
USER node
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

2. In the task definition `volumes` section, define a volume with the name `application_logs`.

```
  "volumes": [
    {
      "name": "application_logs"
    }
  ]
```

3. In the `containerDefinitions` section, create the application container definitions. This is so they mount the storage. The `containerPath` value must match the absolute path that's specified in the `VOLUME` directive from the Dockerfile.

```
  "containerDefinitions": [
    {
      "name": "application1",
```

```
      "image": "my-repo/application",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "application_logs",
          "containerPath": "/var/log/exported"
        }
      ]
    },
    {
      "name": "application2",
      "image": "my-repo/application",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "application_logs",
          "containerPath": "/var/log/exported"
        }
      ]
    }
  ]
```

To provide an empty data volume for a container that's tied to the lifecycle of the host Amazon EC2 instance

For tasks that are hosted on Amazon EC2 instances, you can use bind mounts and have the data tied to the lifecycle of the host Amazon EC2 instance. You can do this by using the `host` parameter and specifying a `sourcePath` value. Any files that exist at the `sourcePath` are presented to the containers at the `containerPath` value. Any files that are written to the `containerPath` value are written to the `sourcePath` value on the host Amazon EC2 instance.

> **Important**
>
> Amazon ECS doesn't sync your storage across Amazon EC2 instances. Tasks that use persistent storage can be placed on any Amazon EC2 instance in your cluster that has available capacity. If your tasks require persistent storage after stopping and restarting, always specify the same Amazon EC2 instance at task launch time with the AWS CLI start-

task command. You can also use Amazon EFS volumes for persistent storage. For more information, see Use Amazon EFS volumes with Amazon ECS.

1. In the task definition `volumes` section, define a bind mount with `name` and `sourcePath` values. In the following example, the host Amazon EC2 instance contains data at `/ecs/webdata` that you want to mount inside the container.

```
"volumes": [
  {
    "name": "webdata",
    "host": {
      "sourcePath": "/ecs/webdata"
    }
  }
]
```

2. In the `containerDefinitions` section, define a container with a `mountPoints` value that references the name of the bind mount and the `containerPath` value to mount the bind mount at on the container.

```
"containerDefinitions": [
  {
    "name": "web",
    "image": "public.ecr.aws/docker/library/nginx:latest",
    "cpu": 99,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webdata",
        "containerPath": "/usr/share/nginx/html"
      }
    ]
  }
```

```
]
```

To mount a defined volume on multiple containers at different locations

You can define a data volume in a task definition and mount that volume at different locations on different containers. For example, your host container has a website data folder at `/data/webroot`. You might want to mount that data volume as read-only on two different web servers that have different document roots.

1. In the task definition `volumes` section, define a data volume with the name `webroot` and the source path `/data/webroot`.

```
"volumes": [
  {
    "name": "webroot",
    "host": {
      "sourcePath": "/data/webroot"
    }
  }
]
```

2. In the `containerDefinitions` section, define a container for each web server with `mountPoints` values that associate the `webroot` volume with the `containerPath` value pointing to the document root for that container.

```
"containerDefinitions": [
  {
    "name": "web-server-1",
    "image": "my-repo/ubuntu-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
```

```
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "web-server-2",
      "image": "my-repo/sles11-apache",
      "cpu": 100,
      "memory": 100,
      "portMappings": [
        {
          "containerPort": 8080,
          "hostPort": 8080
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webroot",
          "containerPath": "/srv/www/htdocs",
          "readOnly": true
        }
      ]
    }
  ]
```

To mount volumes from another container using `volumesFrom`

For tasks hosted on Amazon EC2 instances, you can define one or more volumes on a container, and then use the `volumesFrom` parameter in a different container definition within the same task to mount all of the volumes from the `sourceContainer` at their originally defined mount points. The `volumesFrom` parameter applies to volumes defined in the task definition, and those that are built into the image with a Dockerfile.

1. (Optional) To share a volume that is built into an image, use the `VOLUME` instruction in the Dockerfile. The following example Dockerfile uses an `httpd` image, and then adds a volume and mounts it at `dockerfile_volume` in the Apache document root. It is the folder used by the `httpd` web server.

```
FROM httpd
```

```
VOLUME ["/usr/local/apache2/htdocs/dockerfile_volume"]
```

You can build an image with this Dockerfile and push it to a repository, such as Docker Hub, and use it in your task definition. The example `my-repo/httpd_dockerfile_volume` image that's used in the following steps was built with the preceding Dockerfile.

2. Create a task definition that defines your other volumes and mount points for the containers. In this example `volumes` section, you create an empty volume called `empty`, which the Docker daemon manages. There's also a host volume defined that's called `host_etc`. It exports the `/etc` folder on the host container instance.

```
{
  "family": "test-volumes-from",
  "volumes": [
    {
      "name": "empty",
      "host": {}
    },
    {
      "name": "host_etc",
      "host": {
        "sourcePath": "/etc"
      }
    }
  ],
```

In the container definitions section, create a container that mounts the volumes defined earlier. In this example, the `web` container mounts the `empty` and `host_etc` volumes. This is the container that uses the image built with a volume in the Dockerfile.

```
"containerDefinitions": [
    {
      "name": "web",
      "image": "my-repo/httpd_dockerfile_volume",
      "cpu": 100,
      "memory": 500,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
```

```
      ],
      "mountPoints": [
        {
          "sourceVolume": "empty",
          "containerPath": "/usr/local/apache2/htdocs/empty_volume"
        },
        {
          "sourceVolume": "host_etc",
          "containerPath": "/usr/local/apache2/htdocs/host_etc"
        }
      ],
      "essential": true
    },
```

Create another container that uses `volumesFrom` to mount all of the volumes that are associated with the `web` container. All of the volumes on the `web` container are likewise mounted on the `busybox` container. This includes the volume that's specified in the Dockerfile that was used to build the `my-repo/httpd_dockerfile_volume` image.

```
    {
      "name": "busybox",
      "image": "busybox",
      "volumesFrom": [
        {
          "sourceContainer": "web"
        }
      ],
      "cpu": 100,
      "memory": 500,
      "entryPoint": [
        "sh",
        "-c"
      ],
      "command": [
        "echo $(date) > /usr/local/apache2/htdocs/empty_volume/date && echo $(date)
 > /usr/local/apache2/htdocs/host_etc/date && echo $(date) > /usr/local/apache2/
htdocs/dockerfile_volume/date"
      ],
      "essential": false
    }
  ]
}
```

When this task is run, the two containers mount the volumes, and the `command` in the `busybox` container writes the date and time to a file. This file is called `date` in each of the volume folders. The folders are then visible at the website displayed by the `web` container.

> **Note**
>
> Because the `busybox` container runs a quick command and then exits, it must be set as `"essential": false` in the container definition. Otherwise, it stops the entire task when it exits.

Managing container swap memory space on Amazon ECS

With Amazon ECS, you can control the usage of swap memory space on your Linux-based Amazon EC2 instances at the container level. Using a per-container swap configuration, each container within a task definition can have swap enabled or disabled. For those that have it enabled, the maximum amount of swap space that's used can be limited. For example, latency-critical containers can have swap disabled. In contrast, containers with high transient memory demands can have swap turned on to reduce the chances of out-of-memory errors when the container is under load.

The swap configuration for a container is managed by the following container definition parameters.

`maxSwap`

The total amount of swap memory (in MiB) a container can use. This parameter is translated to the `--memory-swap` option to docker run where the value is the sum of the container memory plus the `maxSwap` value.

If a `maxSwap` value of `0` is specified, the container doesn't use swap. Accepted values are `0` or any positive integer. If the `maxSwap` parameter is omitted, the container uses the swap configuration for the container instance that it's running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

`swappiness`

You can use this to tune a container's memory swappiness behavior. A `swappiness` value of `0` causes swapping to not occur unless required. A `swappiness` value of `100` causes pages to be swapped aggressively. Accepted values are whole numbers between `0` and `100`. If the `swappiness` parameter isn't specified, a default value of `60` is used. If a value isn't specified for `maxSwap`, this parameter is ignored. This parameter maps to the `--memory-swappiness` option to docker run.

In the following example, the JSON syntax is provided.

```
"containerDefinitions": [{
        ...
        "linuxParameters": {
            "maxSwap": integer,
            "swappiness": integer
        },
        ...
}]
```

Considerations

Consider the following when you use a per-container swap configuration.

- Swap space must be enabled and allocated on the Amazon EC2 instance hosting your tasks for the containers to use. By default, the Amazon ECS optimized AMIs do not have swap enabled. You must enable swap on the instance to use this feature. For more information, see Instance Store Swap Volumes in the *Amazon EC2 User Guide* or How do I allocate memory to work as swap space in an Amazon EC2 instance?.
- The swap space container definition parameters are only supported for task definitions that specify the EC2 launch type. These parameters are not supported for task definitions intended only for Amazon ECS on Fargate use.
- This feature is only supported for Linux containers. Windows containers are not supported currently.
- If the `maxSwap` and `swappiness` container definition parameters are omitted from a task definition, each container has a default `swappiness` value of `60`. Moreover, the total swap usage is limited to two times the memory of the container.
- If you're using tasks on Amazon Linux 2023 the `swappiness` parameter isn't supported.

Amazon ECS task definition differences for the Fargate launch type

In order to use Fargate, you must configure your task definition to use the Fargate launch type. There are additional considerations when using Fargate.

Task definition parameters

Tasks that use the Fargate launch type don't support all of the Amazon ECS task definition parameters that are available. Some parameters aren't supported at all, and others behave differently for Fargate tasks.

The following task definition parameters are not valid in Fargate tasks:

- `disableNetworking`
- `dnsSearchDomains`
- `dnsServers`
- `dockerSecurityOptions`
- `extraHosts`
- `gpu`
- `ipcMode`
- `links`
- `placementConstraints`
- `privileged`
- `maxSwap`
- `swappiness`

The following task definition parameters are valid in Fargate tasks, but have limitations that should be noted:

- `linuxParameters` – When specifying Linux-specific options that are applied to the container, for `capabilities` the only capability you can add is `CAP_SYS_PTRACE`. The `devices`, `sharedMemorySize`, and `tmpfs` parameters are not supported. For more information, see [Linux parameters](#).
- `volumes` – Fargate tasks only support bind mount host volumes, so the `dockerVolumeConfiguration` parameter is not supported. For more information, see [Volumes](#).

- `cpu` - For Windows containers on AWS Fargate, the value cannot be less than 1 vCPU.
- `networkConfiguration` - Fargate tasks always use the `awsvpc` network mode.

To ensure that your task definition validates for use with Fargate, you can specify the following when you register the task definition:

- In the AWS Management Console, for the **Requires Compatibilities** field, specify `FARGATE`.
- In the AWS CLI, specify the `--requires-compatibilities` option.
- In the Amazon ECS API, specify the `requiresCompatibilities` flag.

Operating Systems and architectures

When you configure a task and container definition for AWS Fargate, you must specify the Operating System that the container runs. The following Operating Systems are supported for AWS Fargate:

- Amazon Linux 2

> **Note**
>
> Linux containers use only the kernel and kernel configuration from the host Operating System. For example, the kernel configuration includes the `sysctl` system controls. A Linux container image can be made from a base image that contains the files and programs from any Linux distribution. If the CPU architecture matches, you can run containers from any Linux container image on any Operating System.

- Windows Server 2019 Full
- Windows Server 2019 Core
- Windows Server 2022 Full
- Windows Server 2022 Core

When you run Windows containers on AWS Fargate, you must have the X86_64 CPU architecture.

When you run Linux containers on AWS Fargate, you can use the X86_64 CPU architecture, or the ARM64 architecture for your ARM-based applications. For more information, see the section called "Task definitions for 64-bit ARM workloads".

Task CPU and memory

Amazon ECS task definitions for AWS Fargate require that you specify CPU and memory at the task level. Most use cases are satisfied by only specifying these resources at the task level. The table below shows the valid combinations of task-level CPU and memory. You can specify memory values in the task definition as a string in MiB or GB. For example, you can specify a memory value either as `3072` in MiB or `3 GB` in GB. You can specify CPU values in the JSON file as a string in CPU units or virtual CPUs (vCPUs). For example, you can specify a CPU value either as `1024` in CPU units or `1 vCPU` in vCPUs.

CPU value	Memory value	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) **Note** This option requires Linux platform `1.4.0` or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU)	Between 32 GB and 120 GB in 8 GB increments	Linux

CPU value	Memory value	Operating systems supported for AWS Fargate
Note This option requires Linux platform `1.4.0` or later.		

Task networking

Amazon ECS tasks for AWS Fargate require the `awsvpc` network mode, which provides each task with an elastic network interface. When you run a task or create a service with this network mode, you must specify one or more subnets to attach the network interface and one or more security groups to apply to the network interface.

If you are using public subnets, decide whether to provide a public IP address for the network interface. For a Fargate task in a public subnet to pull container images, a public IP address needs to be assigned to the task's elastic network interface, with a route to the internet or a NAT gateway that can route requests to the internet. For a Fargate task in a private subnet to pull container images, you need a NAT gateway in the subnet to route requests to the internet. When you host your container images in Amazon ECR, you can configure Amazon ECR to use an interface VPC endpoint. In this case, the task's private IPv4 address is used for the image pull. For more information about Amazon ECR interface endpoints, see [Amazon ECR interface VPC endpoints (AWS PrivateLink)] in the *Amazon Elastic Container Registry User Guide*.

The following is an example of the `networkConfiguration` section for a Fargate service:

```
"networkConfiguration": {
   "awsvpcConfiguration": {
      "assignPublicIp": "ENABLED",
      "securityGroups": [ "sg-12345678" ],
      "subnets": [ "subnet-12345678" ]
   }
}
```

Task resource limits

Amazon ECS task definitions for Linux containers on AWS Fargate support the `ulimits` parameter to define the resource limits to set for a container.

Amazon ECS task definitions for Windows on AWS Fargate do not support the `ulimits` parameter to define the resource limits to set for a container.

Amazon ECS tasks hosted on Fargate use the default resource limit values set by the operating system with the exception of the `nofile` resource limit parameter. The `nofile` resource limit sets a restriction on the number of open files that a container can use. On Fargate, the default `nofile` soft limit is `65535` and hard limit is `65535`. You can set the values of both limits up to `1048576`.

The following is an example task definition snippet that shows how to define a custom `nofile` limit that has been doubled:

```
"ulimits": [
    {
       "name": "nofile",
       "softLimit": 2048,
       "hardLimit": 8192
    }
]
```

For more information on the other resource limits that can be adjusted, see Resource limits.

Logging

Event logging

Amazon ECS logs the actions that it takes to EventBridge. You can use Amazon ECS events for EventBridge to receive near real-time notifications regarding the current state of your Amazon ECS clusters, services, and tasks. Additionally, you can automate actions to respond to these events. For more information, see Automate responses to Amazon ECS errors using EventBridge.

Task lifecycle logging

Tasks that run on Fargate publish timestamps to track the task through the states of the task lifecycle. You can see the timestamps in the task details in the AWS Management Console and by describing the task in the AWS CLI and SDKs. For example, you can use the timestamps to evaluate how much time the task spent downloading the container images and decide if you should optimize the container image size, or use Seekable OCI indexes. For more information about container image practices, see Best practices for Amazon ECS container images.

Application logging

Amazon ECS task definitions for AWS Fargate support the `awslogs`, `splunk`, and `awsfirelens` log drivers for the log configuration.

The `awslogs` log driver configures your Fargate tasks to send log information to Amazon CloudWatch Logs. The following shows a snippet of a task definition where the `awslogs` log driver is configured:

```
"logConfiguration": {
   "logDriver": "awslogs",
   "options": {
      "awslogs-group" : "/ecs/fargate-task-definition",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
   }
}
```

For more information about using the `awslogs` log driver in a task definition to send your container logs to CloudWatch Logs, see Send Amazon ECS logs to CloudWatch .

For more information about the `awsfirelens` log driver in a task definition, see Send Amazon ECS logs to an AWS service or AWS Partner.

For more information about using the `splunk` log driver in a task definition, see `splunk` log driver.

Task storage

For Amazon ECS tasks hosted on Fargate, the following storage types are supported:

- Amazon EBS volumes provide cost-effective, durable, high-performance block storage for data-intensive containerized workloads. For more information, see Use Amazon EBS volumes with Amazon ECS.
- Amazon EFS volumes for persistent storage. For more information, see Use Amazon EFS volumes with Amazon ECS.
- Bind mounts for ephemeral storage. For more information, see Use bind mounts with Amazon ECS.

Lazy loading container images using Seekable OCI (SOCI)

Amazon ECS tasks on Fargate that use Linux platform version `1.4.0` can use Seekable OCI (SOCI) to help start tasks faster. With SOCI, containers only spend a few seconds on the image pull before they can start, providing time for environment setup and application instantiation while the image is downloaded in the background. This is called *lazy loading*. When Fargate starts an Amazon ECS task, Fargate automatically detects if a SOCI index exists for an image in the task and starts the container without waiting for the entire image to be downloaded.

For containers that run without SOCI indexes, container images are downloaded completely before the container is started. This behavior is the same on all other platform versions of Fargate and on the Amazon ECS-optimized AMI on Amazon EC2 instances.

Seekable OCI (SOCI) is an open source technology developed by AWS that can launch containers faster by lazily loading the container image. SOCI works by creating an index (SOCI Index) of the files within an existing container image. This index helps to launch containers faster, providing the capability to extract an individual file from a container image before downloading the entire image. The SOCI index must be stored as an artifact in the same repository as the image within the container registry. You should only use SOCI indices from trusted sources, as the index is the authoritative source for the contents of the image. For more information, see Introducing Seekable OCI for lazy loading container images.

Customers looking to use SOCI will only be able to use the SOCI index manifest v2. Existing customers that have previously used SOCI on Fargate can continue to use the SOCI Index Manifest v1, however we strongly advise those customers migrate to SOCI Index Manifest v2. The SOCI Index Manifest v2 creates an explicit relationship between container images and their SOCI indexes to ensure consistent deployments.

Considerations

If you want Fargate to use a SOCI index to lazily load container images in a task, consider the following:

- Only tasks that run on Linux platform version `1.4.0` can use SOCI indexes. Tasks that run Windows containers on Fargate aren't supported.
- Tasks that run on X86_64 or ARM64 CPU architecture are supported.
- Container images in the task definition must be stored in a compatible image registry. The following lists the compatible registries:

  - Amazon ECR private registries.
- Only container images that use gzip compression, or are not compressed are supported. Container images that use zstd compression aren't supported.
- For SOCI Index Manifest v2, generating a SOCI index manifest modifies the container image manifest as we add an annotation for the SOCI index. This results in a new container image digest. The contents of the container images filesystem layers do not change.
- For SOCI Index Manifest v2, when the container image has already been stored in the container image repository, after a SOCI index has been generated, you need to repush the container image. Re-pushing a container image will not increase storage costs by duplicating the filesystem layers, it only uploads a new manifest file.
- We recommend that you try lazy loading with container images greater than 250 MiB compressed in size. You are less likely to see a reduction in the time to load smaller images.
- Because lazy loading can change how long your tasks take to start, you might need to change various timeouts like the health check grace period for Elastic Load Balancing.
- If you want to prevent a container image from being lazy loaded, the container image will need to be repushed without a SOCI index attached.

Creating a Seekable OCI index

For a container image to be lazy loaded it needs a SOCI index (a metadata file) created and stored in the container image repository along side the container image. To create and push a SOCI index you can use the open source [soci-snapshotter CLI tool] on GitHub. Or, you can deploy the CloudFormation AWS SOCI Index Builder. This is a serverless solution that automatically creates and pushes a SOCI index when a container image is pushed to Amazon ECR. For more information about the solution and the installation steps, see [CloudFormation AWS SOCI Index Builder] on GitHub. The CloudFormation AWS SOCI Index Builder is a way to automate getting started with SOCI, while the open source soci tool has more flexibility around index generation and the ability to integrate index generation in your continuous integration and continuous delivery (CI/CD) pipelines.

> **Note**
>
> For the SOCI index to be created for an image, the image must exist in the containerd image store on the computer running `soci-snapshotter`. If the image is in the Docker image store, the image can't be found.

Verifying that a task used lazy loading

To verify that a task was lazily loaded using SOCI, check the task metadata endpoint from inside the task. When you query the task metadata endpoint version 4, there is a `Snapshotter` field in the default path for the container that you are querying from. Additionally, there are `Snapshotter` fields for each container in the `/task` path. The default value for this field is `overlayfs`, and this field is set to `soci` if SOCI is used. To verify that a container image has a SOCI Index Manifest v2 attached you can retrieve the Image Index from Amazon ECR using the AWS CLI.

```
IMAGE_REPOSITORY=r
IMAGE_TAG=latest

aws ecr batch-get-image \
    --repository-name=$IMAGE_REPOSITORY \
    --image-ids imageTag=$IMAGE_TAG \
    --query 'images[0].imageManifest' --output text | jq -r '.manifests[] |
 select(.artifactType=="application/vnd.amazon.soci.index.v2+json")'
```

To verify that a container image has a SOCI Index Manifest v1 attached you can use the OCI Referrers API.

```
ACCOUNT_ID=111222333444
AWS_REGION=us-east-1
IMAGE_REPOSITORY=nginx-demo
IMAGE_TAG=latest
IMAGE_DIGEST=$(aws ecr describe-images --repository-name $IMAGE_REPOSITORY --image-ids
 imageTag=$IMAGE_TAG --query 'imageDetails[0].imageDigest' --output text)
ECR_PASSWORD=$(aws ecr get-login-password)

curl \
    --silent \
    --user AWS:$ECR_PASSWORD \
    https://$ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/v2/$IMAGE_REPOSITORY/
referrers/$IMAGE_DIGEST?artifactType=application%2Fvnd.amazon.soci.index.v1%2Bjson | jq
 -r '.'
```

Amazon ECS task definition differences for EC2 instances running Windows

Tasks that run on EC2 Windows instances don't support all of the Amazon ECS task definition parameters that are available. Some parameters aren't supported at all, and others behave differently.

The following task definition parameters aren't supported for Amazon EC2 Windows task definitions:

- `containerDefinitions`
  - `disableNetworking`
  - `dnsServers`
  - `dnsSearchDomains`
  - `extraHosts`
  - `links`
  - `linuxParameters`
  - `privileged`
  - `readonlyRootFilesystem`
  - `user`
  - `ulimits`
- `volumes`
  - `dockerVolumeConfiguration`
- `cpu`

  We recommend specifying container-level CPU for Windows containers.
- `memory`

  We recommend specifying container-level memory for Windows containers.
- `proxyConfiguration`
- `ipcMode`
- `pidMode`
- `taskRoleArn`

The IAM roles for tasks on EC2 Windows instances features requires additional configuration, but much of this configuration is similar to configuring IAM roles for tasks on Linux container instances. For more information see the section called " Amazon EC2 Windows instance additional configuration".

Creating an Amazon ECS task definition using the console

You create a task definition so that you can define the application that you run as a task or service.

When you create a task definition for the external launch type, you need to create the task definition using JSON editor and set the `requireCapabilities` parameter to `EXTERNAL`.

You can create a task definition by using the console experience, or by specifying a JSON file.

JSON validation

The Amazon ECS console JSON editor validates the following in the JSON file:

- The file is a valid JSON file.
- The file doesn't contain any extraneous keys.
- The file contains the `familyName` parameter.
- There is at least one entry under `containerDefinitions`.

AWS CloudFormation stacks

The following behavior applies to task definitions that were created in the new Amazon ECS console before January 12, 2023.

When you create a task definition, the Amazon ECS console automatically creates a CloudFormation stack that has a name that begins with `ECS-Console-V2-TaskDefinition-`. If you used the AWS CLI or an AWS SDK to deregister the task definition, then you must manually delete the task definition stack. For more information, see Deleting a stack in the *AWS CloudFormation User Guide*.

Task definitions created after January 12, 2023, do not have a CloudFormation stack automatically created for them.

Procedure

Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. On the **Create new task definition** menu, choose **Create new task definition**.
4. For **Task definition family**, specify a unique name for the task definition.
5. For **Launch type**, choose the application environment. The console default is **AWS Fargate** (which is serverless). Amazon ECS uses this value to perform validation to ensure that the task definition parameters are valid for the infrastructure type.
6. For **Operating system/Architecture**, choose the operating system and CPU architecture for the task.

   To run your task on a 64-bit ARM architecture, choose **Linux/ARM64**. For more information, see the section called "Runtime platform".

   To run your **AWS Fargate** tasks on Windows containers, choose a supported Windows operating system. For more information, see the section called "Operating Systems and architectures".
7. For **Task size**, choose the CPU and memory values to reserve for the task. The CPU value is specified as vCPUs and memory is specified as GB.

   For tasks hosted on Fargate, the following table shows the valid CPU and memory combinations.

CPU value	Memory value	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows

CPU value	Memory value	Operating systems supported for AWS Fargate
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) **Note** This option requires Linux platform `1.4.0` or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU) **Note** This option requires Linux platform `1.4.0` or later.	Between 32 GB and 120 GB in 8 GB increments	Linux

For tasks that use the EC2 or external launch type, the supported task CPU values are between 128 CPU units (0.125 vCPUs) and 196608 CPU units (192 vCPUs).

To specify the memory value in GB, enter **GB** after the value. For example, to set the **Memory value** to 3GB, enter **3GB**.

> **Note**
>
> Task-level CPU and memory parameters are ignored for Windows containers.

8. For **Network mode**, choose the network mode to use. The default is **awsvpc** mode. For more information, see Amazon ECS task networking.

   If you choose **bridge**, under **Port mappings**, for **Host port**, enter the port number on the container instance to reserve for your container.

9. (Optional) Expand the **Task roles** section to configure the AWS Identity and Access Management (IAM) roles for the task:

   a. For **Task role**, choose the IAM role to assign to the task. A task IAM role provides permissions for the containers in a task to call AWS API operations.

   b. For **Task execution role**, choose the role.

      For information about when to use a task execution role, see the section called “Task execution IAM role”. If you don't need the role, choose **None**.

10. (Optional) Expand the **Task placement** section to add placement contraints. Task placement constraints allow you to filter the container instances used for the placement of your tasks using built-in or custom attributes.

11. (Optional) Expand the **Fault injection** section to enable fault injection. Fault injection lets you test how your application responds to certain impairment scenarios.

12. For each container to define in your task definition, complete the following steps.

    a. For **Name**, enter a name for the container.

    b. For **Image URI**, enter the image to use to start a container. Images in the Amazon ECR Public Gallery registry can be specified by using the Amazon ECR Public registry name only. For example, if `public.ecr.aws/ecs/amazon-ecs-agent:latest` is specified, the Amazon Linux container hosted on the Amazon ECR Public Gallery is used. For all other repositories, specify the repository by using either the `repository-url/image:tag` or `repository-url/image@digest` formats.

    c. If your image is in a private registry outside of Amazon ECR, under **Private registry**, turn on **Private registry authentication**. Then, in **Secrets Manager ARN or name**, enter the Amazon Resource Name (ARN) of the secret.

    d. For **Essential container**, if your task definition has two or more containers defined, you can specify whether the container should be considered essential. When a container is marked as **Essential**, if that container stops, then the task is stopped. Each task definition must contain at least one essential container.

e. A port mapping allows the container to access ports on the host to send or receive traffic. Under **Port mappings**, do one of the following:

   - When you use the **awsvpc** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container.
   - When you use the **bridge** network mode, for **Container port** and **Protocol**, choose the port mapping to use for the container.

   Choose **Add more port mappings** to specify additional container port mappings.

f. To give the container read-only access to its root file system, for **Read only root file system**, select **Read only**.

g. (Optional) To define the container-level CPU, GPU, and memory limits that are different from task-level values, under **Resource allocation limits**, do the following:

   - For **CPU**, enter the number of CPU units that the Amazon ECS container agent reserves for the container.
   - For **GPU**, enter the number of GPU units for the container instance.

     An Amazon EC2 instance with GPU support has 1 GPU unit for every GPU. For more information, see the section called "Task definitions for GPU workloads".
   - For **Memory hard limit**, enter the amount of memory, in GB, to present to the container. If the container attempts to exceed the hard limit, the container stops.
   - The Docker 20.10.0 or later daemon reserves a minimum of 6 mebibytes (MiB) of memory for a container, so don't specify fewer than 6 MiB of memory for your containers.

     The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container, so don't specify fewer than 4 MiB of memory for your containers.
   - For **Memory soft limit**, enter the soft limit (in GB) of memory to reserve for the container.

     When system memory is under contention, Docker attempts to keep the container memory to this soft limit. If you don't specify task-level memory, you must specify a non-zero integer for one or both of **Memory hard limit** and **Memory soft limit**. If you specify both, **Memory hard limit** must be greater than **Memory soft limit**.

     This feature is not supported on Windows containers.

h. (Optional) Expand the **Environment variables** section to specify environment variables to inject into the container. You can specify environment variables either individually by using key-value pairs or in bulk by specifying an environment variable file that's hosted in an Amazon S3 bucket. For information about how to format an environment variable file, see Pass an individual environment variable to an Amazon ECS container.

   When you specify an environment variable for secret storage, for **Key**, enter the secret name. Then for **ValueFrom**, enter the full ARN of the Systems Manager Parameter Store secret or Secrets Manager secret

i. (Optional) Select the **Use log collection** option to specify a log configuration. For each available log driver, there are log driver options to specify. The default option sends container logs to Amazon CloudWatch Logs. The other log driver options are configured by using AWS FireLens. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.

   The following describes each container log destination in more detail.

   - **Amazon CloudWatch** – Configure the task to send container logs to CloudWatch Logs. The default log driver options are provided, which create a CloudWatch log group on your behalf. To specify a different log group name, change the driver option values.
   - **Export logs to Splunk** – Configure the task to send container logs to the Splunk driver that sends the logs to a remote service. You must enter the URL to your Splunk web service. The Splunk token is specified as a secret option because it can be treated as sensitive data.
   - **Export logs to Amazon Data Firehose** – Configure the task to send container logs to Firehose. The default log driver options are provided, which sends log to an Firehose delivery stream. To specify a different delivery stream name, change the driver option values.
   - **Export logs to Amazon Kinesis Data Streams** – Configure the task to send container logs to Kinesis Data Streams. The default log driver options are provided, which send logs to a Kinesis Data Streams stream. To specify a different stream name, change the driver option values.
   - **Export logs to Amazon OpenSearch Service** – Configure the task to send container logs to an OpenSearch Service domain. The log driver options must be provided.

- **Export logs to Amazon S3** – Configure the task to send container logs to an Amazon S3 bucket. The default log driver options are provided, but you must specify a valid Amazon S3 bucket name.

j. (Optional) Configure additional container parameters.

To configure this option	Do this	
Restart policy These options define a restart policy to restart a container when it exits.	Expand **Restart policy**, and then configure the following items: • To enable a restart policy for the container, turn on **Enable Restart policy**. • For **Ignored exit codes**, specify a comma-separated list of integer container exit codes. If the container exits with any of the specified exit codes, Amazon ECS will not try to restart the container. If nothing is specified, Amazon ECS will not ignore any exit codes. • For **Attempt reset period**, specify an integer period of time, in seconds, that the container must run for before a restart can be attempted in the event of an exit. Amazon ECS can attempt to restart a container only	

To configure this option	Do this	
	once every **Attempt reset period** seconds. If nothing is specified, the container must run for 300 seconds before a restart can be attempted.	

To configure this option	Do this	
HealthCheck These are the commands that determine if a container is healthy. For more information, see [Determine Amazon ECS task health using container health checks]().	Expand **HealthCheck**, and then configure the following items: • For **Command**, enter a comma-separated list of commands. You can start the commands with `CMD` to run the command arguments directly, or `CMD-SHELL` to run the command with the container's default shell. If neither is specified, `CMD` is used. • For **Interval**, enter the number of seconds between each health check. The valid values are between 5 and 30. • For **Timeout**, enter the period of time (in seconds) to wait for a health check to succeed before it's considered a failure. The valid values are between 2 and 60. • For **Start period**, enter the period of time (in	

To configure this option	Do this
	seconds) to wait for a container to bootstrap before the health check commands run. The valid values are between 0 and 300. • For **Retries**, enter the number of times to retry the health check commands when there is a failure. The valid values are between 1 and 10.
Startup dependency ordering This option defines dependencies for container startup and shutdown. A container can contain multiple dependencies.	Expand **Startup dependency ordering**, and then configure the following: a. Choose **Add container dependency**. b. For **Container**, choose the container. c. For **Condition**, choose the startup dependency condition. To add an additional dependency, choose **Add container dependency**.

To configure this option	Do this	
Container timeouts These options determine when to start and stop a container.	Expand **Container timeouts**, and then configure the following: • To configure the time to wait before giving up on resolving dependencies for a container, for **Start timeout**, enter the number of seconds. • To configure the time to wait before the container is stopped if it doesn't exit normally on its own, for **Stop timeout**, enter the number of seconds.	

To configure this option	Do this	
Container network settings These options determine whether to use networkin g within a container.	Expand **Container network settings**, and then configure the fo llowing: • To disable container networking, select **Turn off networking**. • To configure DNS server IP addresses that are presented to the container, in **DNS servers**, enter the IP address of each server on a separate line. • To configure DNS domains to search non-fully-qualifi ed host names that are presented to the container, in **DNS search domains**, enter each domain on a sepa rate line. The pattern is `^[a-zA-Z0-9-.]{0,253}[a-zA-Z0-9]$` . • To configure the container host name, in **Host name**, enter	

To configure this option	Do this	
	the container goat name. • To add hostnames and IP address mappings that are appended to the `/etc/hosts` file on the container, choose **Add extra host**, and then for **Hostname** and **IP address**, enter the host name and IP address.	

To configure this option	Do this	
Docker configuration These override the values in the Dockerfile.	Expand **Docker configuration**, and then configure the following items: • For **Command**, enter an executable command for a container. This parameter maps to `Cmd` in the [Create a container](#) section of the Docker Remote API and the `COMMAND` option to `docker run`. This parameter overrides the `CMD` instruction in a [Dockerfile](#). • For **Entry point**, enter the Docker ENTRYPOINT that is passed to the container. This parameter maps to `Entrypoint` in the [Create a container](#) section of the Docker Remote API and the `--entrypoint` option to `docker run`. This parameter overrides	

To configure this option	Do this	
	the `ENTRYPOINT` instruction in a Dockerfile. • For **Working directory**, enter the directory that the container will run any entry point and command instructions provided. This parameter maps to `WorkingDir` in the Create a container section of the Docker Remote API and the `--workdir` option to `docker run`. This parameter overrides the `WORKDIR` instruction in a Dockerfile.	

To configure this option	Do this
Resource limits (Ulimits) These values overwrite the default resource quota setting for the operating system. This parameter maps to `Ulimits` in the [Create a container](#) section of the [Docker Remote API](#) and the `--ulimit` option to [docker run](#).	Expand **Resource limits (ulimits)**, and then choose **Add ulimit**. For **Limit name**, choose the limit. Then, for **Soft limit** and **Hard limit**, enter the values. To add additional ulimits, choose **Add ulimit**.
Docker labels This option adds metadata to your container. This parameter maps to `Labels` in the [Create a container](#) section of the [Docker Remote API](#) and the `--label` option to [docker run](#).	Expand **Docker labels**, choose **Add key value pair**, and then enter the **Key** and **Value**. To add additional Docker labels, choose **Add key value pair**.

k. (Optional) Choose **Add more containers** to add additional containers to the task definition.

13. (Optional) The **Storage** section is used to expand the amount of ephemeral storage for tasks hosted on Fargate. You can also use this section to add a data volume configuration for the task.

    - To expand the available ephemeral storage beyond the default value of 20 gibibytes (GiB) for your Fargate tasks, for **Amount**, enter a value up to 200 GiB.

14. (Optional) To add a data volume configuration for the task definition, choose **Add volume**, and then follow these steps.

    a. For **Volume name**, enter a name for the data volume. The data volume name is used when creating a container mount point.

    b. For **Volume configuration**, select whether you want to configure your volume when creating the task definition or during deployment.

    > **Note**
    >
    > Volumes that can be configured when creating a task definition include Bind mount, Docker, Amazon EFS, and Amazon FSx for Windows File Server. Volumes that can be configured at deployment when running a task, or when creating or updating a service include Amazon EBS.

    c. For **Volume type**, select a volume type compatible with the configuration type that you selected, and then configure the volume type.

Volume type	Steps	
Bind mount	a. Choose **Add mount point**, and then configure the following: • For **Container**, choose the container for the mount point. • For **Source volume**, choose the data volume to mount to the container. • For **Container path**, enter the path on the container to mount the volume. • For **Read only**, select whether the container has read-only access to the volume. b. To add additional mount points, **Add mount point**.	

Volume type	Steps	
EFS	a. For **File system ID**, choose the Amazon EFS file system ID. b. (Optional) For **Root directory**, enter the directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume is used. If you plan to use an EFS access point, leave this field blank. c. (Optional) For **Access point**, choose the access point ID to use. d. (Optional) To encrypt the data between the Amazon EFS file system and the Amazon ECS host or to use the task execution role when mounting the volume, choose **Advanced configurations**, and then configure the following:	

Volume type	Steps	
	• To encrypt the data between the Amazon EFS file system and the Amazon ECS host, select **Transit encryption**, and then for **Port**, enter the port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you don't specify a transit encryption port, it uses the port selection strategy that the Amazon EFS mount helper uses. For more information, see [EFS Mount Helper] in the *Amazon Elastic File System User Guide*. • To use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system, select **IAM authorization**. e.	

Volume type	Steps	
	Choose **Add mount point**, and then configure the following: • For **Container**, choose the container for the mount point. • For **Source volume**, choose the data volume to mount to the container. • For **Container path**, enter the path on the container to mount the volume. • For **Read only**, select whether the container has read-only access to the volume. f. To add additional mount points, **Add mount point**.	

Volume type	Steps	
Docker	a. For **Driver**, enter the Docker volume configuration. Windows containers support only the use of the **local** driver. To use bind mounts, specify a host. b. For **Scope**, choose the volume lifecycle. • To have the lifecycle last when the task starts and stops, choose **Task**. • To have the volume persist after the task stops, choose **Shared**. c. Choose **Add mount point**, and then configure the following: • For **Container**, choose the container for the mount point. • For **Source volume**, choose the data volume to mount to the container. •	

Volume type	Steps	
	For **Container path**, enter the path on the container to mount the volume. • For **Read only**, select whether the container has read-only access to the volume. d. To add additional mount points, **Add mount point**.	

Volume type	Steps	
FSx for Windows File Server	a. For **File system ID**, choose the FSx for Windows File Server file system ID. b. For **Root directory**, enter the directory, enter the directory within the FSx for Windows File Server file system to mount as the root directory inside the host. c. For **Credential parameter**, choose how the credentials are stored. • To use AWS Secrets Manager, enter the Amazon Resource Name (ARN) of a Secrets Manager secret. • To use AWS Systems Manager, enter the Amazon Resource Name (ARN) of a Systems Manager parameter. d. For **Domain**, enter the fully qualified domain name that's hosted by an AWS Directory	

Volume type	Steps	
	Service for Microsoft Active Directory (AWS Managed Microsoft AD) directory or a self -hosted EC2 Active Directory. e. Choose **Add mount point**, and then configure the following: • For **Container**, choose the container for the mount point. • For **Source volume**, choose the data volume to mount to the container. • For **Container path**, enter the path on the container to mount the volume. • For **Read only**, select whether the container has read-only access to the volume. f. To add additional mount points, **Add mount point**.	

Volume type	Steps	
Amazon EBS	a. Choose **Add mount point**, and then configure the following: • For **Container**, choose the container for the mount point. • For **Source volume**, choose the data volume to mount to the container. • For **Container path**, enter the path on the container to mount the volume. • For **Read only**, select whether the container has read-only access to the volume. b. To add additional mount points, **Add mount point**.	

15. To add a volume from another container, choose **Add volume from**, and then configure the following:

   - For **Container**, choose the container.
   - For **Source**, choose the container which has the volume you want to mount.
   - For **Read only**, select whether the container has read-only access to the volume.

16. (Optional) To configure your application trace and metric collection settings by using the AWS Distro for OpenTelemetry integration, expand **Monitoring**, and then select **Use metric collection** to collect and send metrics for your tasks to either Amazon CloudWatch or Amazon Managed Service for Prometheus. When this option is selected, Amazon ECS creates an AWS Distro for OpenTelemetry container sidecar that is preconfigured to send the application metrics. For more information, see Correlate Amazon ECS application performance using application metrics.

    a. When **Amazon CloudWatch** is selected, your custom application metrics are routed to CloudWatch as custom metrics. For more information, see Exporting application metrics to Amazon CloudWatch.

    > **Important**
    >
    > When exporting application metrics to Amazon CloudWatch, your task definition requires a task IAM role with the required permissions. For more information, see Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon CloudWatch.

    b. When you select **Amazon Managed Service for Prometheus (Prometheus libraries instrumentation)**, your task-level CPU, memory, network, and storage metrics and your custom application metrics are routed to Amazon Managed Service for Prometheus. For **Workspace remote write endpoint**, enter the remote write endpoint URL for your Prometheus workspace. For **Scraping target**, enter the host and port the AWS Distro for OpenTelemetry collector can use to scrape for metrics data. For more information, see Exporting application metrics to Amazon Managed Service for Prometheus.

    > **Important**
    >
    > When exporting application metrics to Amazon Managed Service for Prometheus, your task definition requires a task IAM role with the required permissions. For more information, see Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus.

    c. When you select **Amazon Managed Service for Prometheus (OpenTelemetry instrumentation)**, your task-level CPU, memory, network, and storage metrics and your custom application metrics are routed to Amazon Managed Service for Prometheus. For **Workspace remote write endpoint**, enter the remote write endpoint URL for your Prometheus workspace. For more information, see [Exporting application metrics to Amazon Managed Service for Prometheus](#).

> **Important**
>
> When exporting application metrics to Amazon Managed Service for Prometheus, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus](#).

17. (Optional) Expand the **Tags** section to add tags, as key-value pairs, to the task definition.

    - [Add a tag] Choose **Add tag**, and then do the following:
      - For **Key**, enter the key name.
      - For **Value**, enter the key value.
    - [Remove a tag] Next to the tag, choose **Remove tag**.

18. Choose **Create** to register the task definition.

Amazon ECS console JSON editor

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. On the **Create new task definition** menu, choose **Create new task definition with JSON**.
4. In the JSON editor box, edit your JSON file,

   The JSON must pass the validation checks specified in the section called “JSON validation”.
5. Choose **Create**.

Updating an Amazon ECS task definition using the console

A *task definition revision* is a copy of the current task definition with the new parameter values replacing the existing ones. All parameters that you do not modify are in the new revision.

To update a task definition, create a task definition revision. If the task definition is used in a service, you must update that service to use the updated task definition.

When you create a revision, you can modify the following container properties and environment properties.

- Container image URI
- Port mappings
- Environment variables
- Task size
- Container size
- Task role
- Task execution role
- Volumes and container mount points
- Private registry

JSON validation

The Amazon ECS console JSON editor validates the following in the JSON file:

- The file is a valid JSON file
- The file does not contain any extraneous keys
- The file contains the `familyName` parameter
- There is at least one entry under `containerDefinitions`

Procedure

Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, choose the Region that contains your task definition.
3. In the navigation pane, choose **Task definitions**.
4. Choose the task definition.

5. Select the task definition revision, and then choose **Create new revision**, **Create new revision**.
6. On the **Create new task definition revision** page, make changes. For example, to change the existing container definitions (such as the container image, memory limits, or port mappings), select the container, and then make the changes.
7. Verify the information, and then choose **Update**.
8. If your task definition is used in a service, update your service with the updated task definition. For more information, see Updating an Amazon ECS service.

Amazon ECS console JSON editor

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new revision**, **Create new revision with JSON**.
4. In the JSON editor box, edit your JSON file,

   The JSON must pass the validation checks specified in the section called "JSON validation".
5. Choose **Create**.

Deregistering an Amazon ECS task definition revision using the console

You can deregister the task definition revision so that it no longer displays in your `ListTaskDefinition` API calls or in the console when you want to run a task or update a service.

When you deregister a task definition revision, it is immediately marked as `INACTIVE`. Existing tasks and services that reference an `INACTIVE` task definition revision continue to run without disruption. Existing services that reference an `INACTIVE` task definition revision can still scale up or down by modifying the service's desired count.

You can't use an `INACTIVE` task definition revision to run new tasks or create new services. You also can't update an existing service to reference an `INACTIVE` task definition revision (even though there may be up to a 10-minute window following deregistration where these restrictions have not yet taken effect).

Note

When you deregister all revisions in a task family, the task definition family is moved to the `INACTIVE` list. Adding a new revision of an `INACTIVE` task definition moves the task definition family back to the `ACTIVE` list.

At this time, `INACTIVE` task definition revisions remain discoverable in your account indefinitely. However, this behavior is subject to change in the future. Therefore, you should not rely on `INACTIVE` task definition revisions persisting beyond the lifecycle of any associated tasks and services.

AWS CloudFormation stacks

The following behavior applies to task definitions that were created in the new Amazon ECS console before January 12, 2023.

When you create a task definition, the Amazon ECS console automatically creates a CloudFormation stack that has a name that begins with `ECS-Console-V2-TaskDefinition-`. If you used the AWS CLI or an AWS SDK to deregister the task definition, then you must manually delete the task definition stack. For more information, see Deleting a stack in the *AWS CloudFormation User Guide*.

Task definitions created after January 12, 2023, do not have a CloudFormation stack automatically created for them.

Procedure

To deregister a new task definition (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, choose the region that contains your task definition.
3. In the navigation pane, choose **Task definitions**.
4. On the **Task definitions** page, choose the task definition family that contains one or more revisions that you want to deregister.
5. On the **task definition Name** page, select the revisions to delete, and then choose **Actions**, **Deregister**.
6. Verify the information in the **Deregister** window, and then choose **Deregister** to finish.

Deleting an Amazon ECS task definition revision using the console

When no longer need a specific task definition revision in Amazon ECS, you can delete the task definition revision.

When you delete a task definition revision, it immediately transitions from the `INACTIVE` to `DELETE_IN_PROGRESS`. Existing tasks and services that reference a `DELETE_IN_PROGRESS` task definition revision continue to run without disruption.

You can't use a `DELETE_IN_PROGRESS` task definition revision to run new tasks or create new services. You also can't update an existing service to reference a `DELETE_IN_PROGRESS` task definition revision.

When you delete all `INACTIVE` task definition revisions, the task definition name is not displayed in the console and not returned in the API. If a task definition revision is in the `DELETE_IN_PROGRESS` state, the task definition name is displayed in the console and returned in the API. The task definition name is retained by Amazon ECS and the revision is incremented the next time you create a task definition with that name.

Amazon ECS resources that can block a deletion

A task definition deletion request will not complete when there are any Amazon ECS resources that depend on the task definition revision. The following resources might prevent a task definition from being deleted:

- Amazon ECS standalone tasks - The task definition is required in order for the task to remain healthy.
- Amazon ECS service tasks - The task definition is required in order for the task to remain healthy.
- Amazon ECS service deployments and task sets - The task definition is required when a scaling event is initiated for an Amazon ECS deployment or task set.

If your task definition remains in the `DELETE_IN_PROGRESS` state, you can use the console, or the AWS CLI to identify, and then stop the resources which block the task definition deletion.

Task definition deletion after the blocked resource is removed

The following rules apply after you remove the resources that block the task definition deletion:

- Amazon ECS tasks - The task definition deletion can take up to 1 hour to complete after the task is stopped.
- Amazon ECS service deployments and task sets - The task definition deletion can take up to 24 hours to complete after the deployment or task set is deleted.

Procedure

To delete task definitions (Amazon ECS console)

You must deregister a task definition revision before you delete it. For more information, see the section called “Deregistering a task definition revision using the console”.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, choose the region that contains your task definition.
3. In the navigation pane, choose **Task definitions**.
4. On the **Task definitions** page, choose the task definition family that contains one or more revisions that you want to delete.
5. On the **Task definition name** page, select the revisions to delete, and then choose **Actions**, **Delete**.

   If **Delete** is unavailable, you must deregister the task definition.
6. Verify the information in the **Delete** confirmation box, and then choose **Delete** to finish.

Amazon ECS task definition use cases

Learn more about how to write task definitions for various AWS services and features.

Depending on your workload, there are certain task definition parameters that need to be set. Also for the EC2 launch type, you have to choose specific instances that are engineered for the workload.

Topics

- Amazon ECS task definitions for GPU workloads
- Amazon ECS task definitions for video transcoding workloads
- Amazon ECS task definitions for AWS Neuron machine learning workloads
- Amazon ECS task definitions for deep learning instances

- Amazon ECS task definitions for 64-bit ARM workloads
- Send Amazon ECS logs to CloudWatch
- Send Amazon ECS logs to an AWS service or AWS Partner
- Using non-AWS container images in Amazon ECS
- Restart individual containers in Amazon ECS tasks with container restart policies
- Pass sensitive data to an Amazon ECS container

Amazon ECS task definitions for GPU workloads

Amazon ECS supports workloads that use GPUs, when you create clusters with container instances that support GPUs. Amazon EC2 GPU-based container instances that use the p2, p3, p5, g3, g4, and g5 instance types provide access to NVIDIA GPUs. For more information, see Linux Accelerated Computing Instances in the *Amazon EC2 Instance Types guide*.

Amazon ECS provides a GPU-optimized AMI that comes with pre-configured NVIDIA kernel drivers and a Docker GPU runtime. For more information, see Amazon ECS-optimized Linux AMIs.

You can designate a number of GPUs in your task definition for task placement consideration at a container level. Amazon ECS schedules to available container instances that support GPUs and pin physical GPUs to proper containers for optimal performance.

The following Amazon EC2 GPU-based instance types are supported. For more information, see Amazon EC2 P2 Instances, Amazon EC2 P3 Instances, Amazon EC2 P4d Instances, Amazon EC2 P5 Instances, Amazon EC2 G3 Instances, Amazon EC2 G4 Instances, Amazon EC2 G5 Instances, Amazon EC2 G6 Instances, and Amazon EC2 G6e Instances.

Instance type	GPUs	GPU memory (GiB)	vCPUs	Memory (GiB)
p3.2xlarge	1	16	8	61
p3.8xlarge	4	64	32	244
p3.16xlarge	8	128	64	488
p3dn.24xlarge	8	256	96	768
p4d.24xlarge	8	320	96	1152

Instance type	GPUs	GPU memory (GiB)	vCPUs	Memory (GiB)
p5.48xlarge	8	640	192	2048
g3s.xlarge	1	8	4	30.5
g3.4xlarge	1	8	16	122
g3.8xlarge	2	16	32	244
g3.16xlarge	4	32	64	488
g4dn.xlarge	1	16	4	16
g4dn.2xlarge	1	16	8	32
g4dn.4xlarge	1	16	16	64
g4dn.8xlarge	1	16	32	128
g4dn.12xlarge	4	64	48	192
g4dn.16xlarge	1	16	64	256
g5.xlarge	1	24	4	16
g5.2xlarge	1	24	8	32
g5.4xlarge	1	24	16	64
g5.8xlarge	1	24	32	128
g5.16xlarge	1	24	64	256
g5.12xlarge	4	96	48	192
g5.24xlarge	4	96	96	384
g5.48xlarge	8	192	192	768
g6.xlarge	1	24	4	16

Instance type	GPUs	GPU memory (GiB)	vCPUs	Memory (GiB)
g6.2xlarge	1	24	8	32
g6.4xlarge	1	24	16	64
g6.8xlarge	1	24	32	128
g6.16.xlarge	1	24	64	256
g6.12xlarge	4	96	48	192
g6.24xlarge	4	96	96	384
g6.48xlarge	8	192	192	768
g6.metal	8	192	192	768
gr6.4xlarge	1	24	16	128
g6e.xlarge	1	48	4	32
g6e.2xlarge	1	48	8	64
g6e.4xlarge	1	48	16	128
g6e.8xlarge	1	48	32	256
g6e16.xlarge	1	48	64	512
g6e12.xlarge	4	192	48	384
g6e24.xlarge	4	192	96	768
g6e48.xlarge	8	384	192	1536
gr6.8xlarge	1	24	32	256

You can retrieve the Amazon Machine Image (AMI) ID for Amazon ECS-optimized AMIs by querying the AWS Systems Manager Parameter Store API. Using this parameter, you don't need to manually look up Amazon ECS-optimized AMI IDs. For more information about the Systems Manager Parameter Store API, see GetParameter. The user that you use must have the `ssm:GetParameter` IAM permission to retrieve the Amazon ECS-optimized AMI metadata.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

Considerations

Note

The support for g2 instance family type has been deprecated.
The p2 instance family type is only supported on versions earlier than `20230912` of the Amazon ECS GPU-optimized AMI. If you need to continue to use p2 instances, see What to do if you need a P2 instance.
In-place updates of the NVIDIA/CUDA drivers on both these instance family types will cause potential GPU workload failures.

We recommend that you consider the following before you begin working with GPUs on Amazon ECS.

- Your clusters can contain a mix of GPU and non-GPU container instances.
- You can run GPU workloads on external instances. When registering an external instance with your cluster, ensure the `--enable-gpu` flag is included on the installation script. For more information, see Registering an external instance to an Amazon ECS cluster.
- You must set `ECS_ENABLE_GPU_SUPPORT` to `true` in your agent configuration file. For more information, see the section called "Container agent configuration".
- When running a task or creating a service, you can use instance type attributes when you configure task placement constraints to determine the container instances the task is to be launched on. By doing this, you can more effectively use your resources. For more information, see How Amazon ECS places tasks on container instances.

  The following example launches a task on a `g4dn.xlarge` container instance in your default cluster.

  ```
  aws ecs run-task --cluster default --task-definition ecs-gpu-task-def \
  ```

```
    --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
 g4dn.xlarge" --region us-east-2
```

- For each container that has a GPU resource requirement that's specified in the container definition, Amazon ECS sets the container runtime to be the NVIDIA container runtime.
- The NVIDIA container runtime requires some environment variables to be set in the container to function properly. For a list of these environment variables, see Specialized Configurations with Docker. Amazon ECS sets the `NVIDIA_VISIBLE_DEVICES` environment variable value to be a list of the GPU device IDs that Amazon ECS assigns to the container. For the other required environment variables, Amazon ECS doesn't set them. So, make sure that your container image sets them or they're set in the container definition.
- The p5 instance type family is supported on version `20230929` and later of the Amazon ECS GPU-optimized AMI.
- The g4 instance type family is supported on version `20230913` and later of the Amazon ECS GPU-optimized AMI. For more information, see Amazon ECS-optimized Linux AMIs. It's not supported in the Create Cluster workflow in the Amazon ECS console. To use these instance types, you must either use the Amazon EC2 console, AWS CLI, or API and manually register the instances to your cluster.
- The p4d.24xlarge instance type only works with CUDA 11 or later.
- The Amazon ECS GPU-optimized AMI has IPv6 enabled, which causes issues when using `yum`. This can be resolved by configuring `yum` to use IPv4 with the following command.

```
echo "ip_resolve=4" >> /etc/yum.conf
```

- When you build a container image that doesn't use the NVIDIA/CUDA base images, you must set the `NVIDIA_DRIVER_CAPABILITIES` container runtime variable to one of the following values:
  - `utility,compute`
  - `all`

  For information about how to set the variable, see Controlling the NVIDIA Container Runtime on the NVIDIA website.
- GPUs are not supported on Windows containers.

Launch a GPU container instance for Amazon ECS

To use a GPU instance on Amazon ECS, you need to create a launch template, a user data file, and launch the instance.

You can then run a task that uses a task definition configured for GPU.

Use a launch template

You can create a launch template.

- Create a launch template that uses the Amazon ECS-optimized GPU AMI ID For the AMI. For information about how to create a launch template, see Create a new launch template using parameters you define in the *Amazon EC2 User Guide*.

  Use the AMI ID from the previous step for the **Amazon Machine image**. For information about how to specify the AMI ID with the Systems Manager parameter, see Specify a Systems Manager parameter in a launch template in the *Amazon EC2 User Guide*.

  Add the following to the **User data** in the launch template. Replace *cluster-name* with the name of your cluster.

  ```
  #!/bin/bash
  echo ECS_CLUSTER=cluster-name >> /etc/ecs/ecs.config;
  echo ECS_ENABLE_GPU_SUPPORT=true >> /etc/ecs/ecs.config
  ```

Use the AWS CLI

You can use the AWS CLI to launch the container instance.

1. Create a file that's called `userdata.toml`. This file is used for the instance user data. Replace *cluster-name* with the name of your cluster.

   ```
   #!/bin/bash
   echo ECS_CLUSTER=cluster-name >> /etc/ecs/ecs.config;
   echo ECS_ENABLE_GPU_SUPPORT=true >> /etc/ecs/ecs.config
   ```

2. Run the following command to get the GPU AMI ID. You use this in the following step.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

3. Run the following command to launch the GPU instance. Remember to replace the following parameters:

   - Replace *subnet* with the ID of the private or public subnet that your instance will launch in.
   - Replace *gpu_ami* with the AMI ID from the previous step.
   - Replace *t3.large* with the instance type that you want to use.
   - Replace *region* with the Region code.

```
aws ec2 run-instances --key-name ecs-gpu-example \
   --subnet-id subnet \
   --image-id gpu_ami \
   --instance-type t3.large \
   --region region \
   --tag-specifications 'ResourceType=instance,Tags=[{Key=GPU,Value=example}]' \
   --user-data file://userdata.toml \
   --iam-instance-profile Name=ecsInstanceRole
```

4. Run the following command to verify that the container instance is registered to the cluster. When you run this command, remember to replace the following parameters:

   - Replace *cluster* with your cluster name.
   - Replace *region* with your Region code.

```
aws ecs list-container-instances --cluster cluster-name --region region
```

Specifying GPUs in an Amazon ECS task definition

To use the GPUs on a container instance and the Docker GPU runtime, make sure that you designate the number of GPUs your container requires in the task definition. As containers that support GPUs are placed, the Amazon ECS container agent pins the desired number of physical GPUs to the appropriate container. The number of GPUs reserved for all containers in a task cannot exceed the number of available GPUs on the container instance the task is launched on. For more information, see Creating an Amazon ECS task definition using the console.

> **Important**
>
> If your GPU requirements aren't specified in the task definition, the task uses the default Docker runtime.

The following shows the JSON format for the GPU requirements in a task definition:

```
{
  "containerDefinitions": [
     {
        ...
        "resourceRequirements" : [
            {
               "type" : "GPU",
               "value" : "2"
            }
        ],
     },
...
}
```

The following example demonstrates the syntax for a Docker container that specifies a GPU requirement. This container uses two GPUs, runs the `nvidia-smi` utility, and then exits.

```
{
  "containerDefinitions": [
    {
      "memory": 80,
      "essential": true,
      "name": "gpu",
      "image": "nvidia/cuda:11.0.3-base",
      "resourceRequirements": [
         {
           "type":"GPU",
           "value": "2"
         }
      ],
      "command": [
        "sh",
        "-c",
        "nvidia-smi"
```

```
      ],
      "cpu": 100
    }
  ],
  "family": "example-ecs-gpu"
}
```

Share GPUs

When you want to share GPUs, you need to configure the following.

1. Remove GPU resource requirements from your task definitions so that Amazon ECS does not reserve any GPUs that should be shared.
2. Add the following user data to your instances when you want to share GPUs. This will make nvidia the default Docker container runtime on the container instance so that all Amazon ECS containers can use the GPUs. For more information see Run commands when you launch an EC2 instance with user data input in the *Amazon EC2 User Guide*.

```
const userData = ec2.UserData.forLinux();
 userData.addCommands(
 'sudo rm /etc/sysconfig/docker',
 'echo DAEMON_MAXFILES=1048576 | sudo tee -a /etc/sysconfig/docker',
 'echo OPTIONS="--default-ulimit nofile=32768:65536 --default-runtime nvidia" |
 sudo tee -a /etc/sysconfig/docker',
 'echo DAEMON_PIDFILE_TIMEOUT=10 | sudo tee -a /etc/sysconfig/docker',
 'sudo systemctl restart docker',
);
```

3. Set the `NVIDIA_VISIBLE_DEVICES` environment variable on your container. You can do this by specifying the environment variable in your task definition. For information on the valid values, see GPU Enumeration on the NVIDIA documentation site.

What to do if you need a P2 instance

If you need to use P2 instance, you can use one of the following options to continue using the instances.

You must modify the instance user data for both options. For more information see Run commands when you launch an EC2 instance with user data input in the *Amazon EC2 User Guide*.

Use the last supported GPU-optimized AMI

You can use the `20230906` version of the GPU-optimized AMI, and add the following to the instance user data.

Replace cluster-name with the name of your cluster.

```
#!/bin/bash
echo "exclude=*nvidia* *cuda*" >> /etc/yum.conf
echo "ECS_CLUSTER=cluster-name" >> /etc/ecs/ecs.config
```

Use the latest GPU-optimized AMI, and update the user data

You can add the following to the instance user data. This uninstalls the Nvidia 535/Cuda12.2 drivers, and then installs the Nvidia 470/Cuda11.4 drivers and fixes the version.

```
#!/bin/bash
yum remove -y cuda-toolkit* nvidia-driver-latest-dkms*
tmpfile=$(mktemp)
cat >$tmpfile <<EOF
[amzn2-nvidia]
name=Amazon Linux 2 Nvidia repository
mirrorlist=\$awsproto://\$amazonlinux.\$awsregion.\$awsdomain/\$releasever/amzn2-
nvidia/latest/\$basearch/mirror.list
priority=20
gpgcheck=1
gpgkey=https://developer.download.nvidia.com/compute/cuda/repos/rhel7/
x86_64/7fa2af80.pub
enabled=1
exclude=libglvnd-*
EOF

mv $tmpfile /etc/yum.repos.d/amzn2-nvidia-tmp.repo
yum install -y system-release-nvidia cuda-toolkit-11-4 nvidia-driver-latest-
dkms-470.182.03
yum install -y libnvidia-container-1.4.0 libnvidia-container-tools-1.4.0 nvidia-
container-runtime-hook-1.4.0 docker-runtime-nvidia-1

echo "exclude=*nvidia* *cuda*" >> /etc/yum.conf
nvidia-smi
```

Create your own P2 compatible GPU-optimized AMI

You can create your own custom Amazon ECS GPU-optimized AMI that is compatible with P2 instances, and then launch P2 instances using the AMI.

1. Run the following command to clone the `amazon-ecs-ami repo`.

```
git clone https://github.com/aws/amazon-ecs-ami
```

2. Set the required Amazon ECS agent and source Amazon Linux AMI versions in `release.auto.pkrvars.hcl` or `overrides.auto.pkrvars.hcl`.
3. Run the following command to build a private P2 compatible EC2 AMI.

   Replace region with the Region with the instance Region.

```
REGION=region make al2keplergpu
```

4. Use the AMI with the following instance user data to connect to the Amazon ECS cluster.

   Replace cluster-name with the name of your cluster.

```
#!/bin/bash
echo "ECS_CLUSTER=cluster-name" >> /etc/ecs/ecs.config
```

Amazon ECS task definitions for video transcoding workloads

To use video transcoding workloads on Amazon ECS, register Amazon EC2 VT1 instances. After you registered these instances, you can run live and pre-rendered video transcoding workloads as tasks on Amazon ECS. Amazon EC2 VT1 instances use Xilinx U30 media transcoding cards to accelerate live and pre-rendered video transcoding workloads.

> **Note**
>
> For instructions on how to run video transcoding workloads in containers other than Amazon ECS, see the Xilinx documentation.

Considerations

Before you begin deploying VT1 on Amazon ECS, consider the following:

- Your clusters can contain a mix of VT1 and non-VT1 instances.
- You need a Linux application that uses Xilinx U30 media transcoding cards with accelerated AVC (H.264) and HEVC (H.265) codecs.

> **Important**
>
> Applications that use other codecs might not have improved performance on VT1 instances.

- Only one transcoding task can run on a U30 card. Each card has two devices that are associated with it. You can run as many transcoding tasks as there are cards for each of your VT1 instance.
- When creating a service or running a standalone task, you can use instance type attributes when configuring task placement constraints. This ensures that the task is launched on the container instance that you specify. Doing so helps ensure that you use your resources effectively and that your tasks for video transcoding workloads are on your VT1 instances. For more information, see How Amazon ECS places tasks on container instances.

  In the following example, a task is run on a `vt1.3xlarge` instance on your `default` cluster.

```
aws ecs run-task \
     --cluster default \
     --task-definition vt1-3xlarge-xffmpeg-processor \
     --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
 vt1.3xlarge"
```

- You configure a container to use the specific U30 card available on the host container instance. You can do this by using the `linuxParameters` parameter and specifying the device details. For more information, see Task definition requirements.

Using a VT1 AMI

You have two options for running an AMI on Amazon EC2 for Amazon ECS container instances. The first option is to use the Xilinx official AMI on the AWS Marketplace. The second option is to build your own AMI from the sample repository.

- Xilinx offers AMIs on the AWS Marketplace.
- Amazon ECS provides a sample repository that you can use to build an AMI for video transcoding workloads. This AMI comes with Xilinx U30 drivers. You can find the repository that contains Packer scripts on GitHub. For more information about Packer, see the Packer documentation.

Task definition requirements

To run video transcoding containers on Amazon ECS, your task definition must contain a video transcoding application that uses the accelerated H.264/AVC and H.265/HEVC codecs. You can build a container image by following the steps on the [Xilinx GitHub](#).

The task definition must be specific to the instance type. The instance types are 3xlarge, 6xlarge, and 24xlarge. You must configure a container to use specific Xilinx U30 devices that are available on the host container instance. You can do so using the `linuxParameters` parameter. The following table details the cards and device SoCs that are specific to each instance type.

Instance Type	vCPUs	RAM (GiB)	U30 accelerator cards	Addressable XCU30 SoC devices	Device Paths
vt1.3xlarge	12	24	1	2	`/dev/dri/renderD128 ,/dev/dri/renderD129`
vt1.6xlarge	24	48	2	4	`/dev/dri/renderD128 ,/dev/dri/renderD129 ,/dev/dri/renderD130 ,/dev/dri/renderD131`
vt1.24xlarge	96	182	8	16	`/dev/dri/renderD128 ,/dev/`

Instance Type	vCPUs	RAM (GiB)	U30 accelerator cards	Addressable XCU30 SoC devices	Device Paths
					dri/renderD129 ,/dev/dri/renderD130 ,/dev/dri/renderD131 ,/dev/dri/renderD132 ,/dev/dri/renderD133 ,/dev/dri/renderD134 ,/dev/dri/renderD135 ,/dev/dri/renderD136 ,/dev/dri/renderD137 ,/dev/dri/renderD138 ,/dev/dri/renderD13

Instance Type	vCPUs	RAM (GiB)	U30 accelerator cards	Addressable XCU30 SoC devices	Device Paths
					9 ,/dev/dri/renderD140 ,/dev/dri/renderD141 ,/dev/dri/renderD142 ,/dev/dri/renderD143

> **Important**
>
> If the task definition lists devices that the EC2 instance doesn't have, the task fails to run. When the task fails, the following error message appears in the stoppedReason: `CannotStartContainerError: Error response from daemon: error gathering device information while adding custom device "/dev/dri/renderD130": no such file or directory`.

Specifying video transcoding in an Amazon ECS task definition

In the following example, the syntax that's used for a task definition of a Linux container on Amazon EC2 is provided. This task definition is for container images that are built following the procedure that's provided in the [Xilinx documentation](). If you use this example, replace `image` with your own image, and copy your video files into the instance in the `/home/ec2-user` directory.

vt1.3xlarge

1. Create a text file that's named `vt1-3xlarge-ffmpeg-linux.json` with the following content.

```
{
    "family": "vt1-3xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.3xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
```

```
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. Register the task definition.

```
aws ecs register-task-definition --family vt1-3xlarge-xffmpeg-processor --cli-
input-json file://vt1-3xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.6xlarge

1. Create a text file that's named `vt1-6xlarge-ffmpeg-linux.json` with the following content.

```
{
    "family": "vt1-6xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
```

```
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == vt1.6xlarge"
        }
    ],
    "containerDefinitions": [
        {
            "entryPoint": [
                "/bin/bash",
                "-c"
            ],
            "command": ["/video/ecs_ffmpeg_wrapper.sh"],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/dri/renderD128",
                        "hostPath": "/dev/dri/renderD128",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD129",
                        "hostPath": "/dev/dri/renderD129",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD130",
                        "hostPath": "/dev/dri/renderD130",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    },
                    {
                        "containerPath": "/dev/dri/renderD131",
                        "hostPath": "/dev/dri/renderD131",
                        "permissions": [
                            "read",
                            "write"
```

```
                        ]
                    }
                ]
            },
            "mountPoints": [
                {
                    "containerPath": "/video",
                    "sourceVolume": "video_file"
                }
            ],
            "cpu": 0,
            "memory": 12000,
            "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
            "essential": true,
            "name": "xilinix-xffmpeg"
        }
    ],
    "volumes": [
        {
            "name": "video_file",
            "host": {"sourcePath": "/home/ec2-user"}
        }
    ]
}
```

2. Register the task definition.

```
aws ecs register-task-definition --family vt1-6xlarge-xffmpeg-processor --cli-
input-json file://vt1-6xlarge-xffmpeg-linux.json --region us-east-1
```

vt1.24xlarge

1. Create a text file that's named `vt1-24xlarge-ffmpeg-linux.json` with the following content.

```
{
    "family": "vt1-24xlarge-xffmpeg-processor",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
```

```
                "expression": "attribute:ecs.os-type == linux"
            },
            {
                "type": "memberOf",
                "expression": "attribute:ecs.instance-type == vt1.24xlarge"
            }
        ],
        "containerDefinitions": [
            {
                "entryPoint": [
                    "/bin/bash",
                    "-c"
                ],
                "command": ["/video/ecs_ffmpeg_wrapper.sh"],
                "linuxParameters": {
                    "devices": [
                        {
                            "containerPath": "/dev/dri/renderD128",
                            "hostPath": "/dev/dri/renderD128",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD129",
                            "hostPath": "/dev/dri/renderD129",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD130",
                            "hostPath": "/dev/dri/renderD130",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD131",
                            "hostPath": "/dev/dri/renderD131",
                            "permissions": [
```

```
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD132",
                            "hostPath": "/dev/dri/renderD132",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD133",
                            "hostPath": "/dev/dri/renderD133",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD134",
                            "hostPath": "/dev/dri/renderD134",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD135",
                            "hostPath": "/dev/dri/renderD135",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD136",
                            "hostPath": "/dev/dri/renderD136",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
```

```
                        {
                            "containerPath": "/dev/dri/renderD137",
                            "hostPath": "/dev/dri/renderD137",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD138",
                            "hostPath": "/dev/dri/renderD138",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD139",
                            "hostPath": "/dev/dri/renderD139",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD140",
                            "hostPath": "/dev/dri/renderD140",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD141",
                            "hostPath": "/dev/dri/renderD141",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD142",
                            "hostPath": "/dev/dri/renderD142",
                            "permissions": [
```

```
                                "read",
                                "write"
                            ]
                        },
                        {
                            "containerPath": "/dev/dri/renderD143",
                            "hostPath": "/dev/dri/renderD143",
                            "permissions": [
                                "read",
                                "write"
                            ]
                        }
                    ]
                },
                "mountPoints": [
                    {
                        "containerPath": "/video",
                        "sourceVolume": "video_file"
                    }
                ],
                "cpu": 0,
                "memory": 12000,
                "image": "0123456789012.dkr.ecr.us-west-2.amazonaws.com/aws/xilinx-
xffmpeg",
                "essential": true,
                "name": "xilinix-xffmpeg"
            }
        ],
        "volumes": [
            {
                "name": "video_file",
                "host": {"sourcePath": "/home/ec2-user"}
            }
        ]
}
```

2. Register the task definition.

```
aws ecs register-task-definition --family vt1-24xlarge-xffmpeg-processor --cli-
input-json file://vt1-24xlarge-xffmpeg-linux.json --region us-east-1
```

Amazon ECS task definitions for AWS Neuron machine learning workloads

You can register Amazon EC2 Trn1, Amazon EC2 Inf1, and Amazon EC2 Inf2 instances to your clusters for machine learning workloads.

Amazon EC2 Trn1 instances are powered by AWS Trainium chips. These instances provide high performance and low cost training for machine learning in the cloud. You can train a machine learning inference model using a machine learning framework with AWS Neuron on a Trn1 instance. Then, you can run the model on a Inf1 instance, or an Inf2 instance to use the acceleration of the AWS Inferentia chips.

The Amazon EC2 Inf1 instances and Inf2 instances are powered by AWS Inferentia chips They provide high performance and lowest cost inference in the cloud.

Machine learning models are deployed to containers using AWS Neuron, which is a specialized Software Developer Kit (SDK). The SDK consists of a compiler, runtime, and profiling tools that optimize the machine learning performance of AWS machine learning chips. AWS Neuron supports popular machine learning frameworks such as TensorFlow, PyTorch, and Apache MXNet.

Considerations

Before you begin deploying Neuron on Amazon ECS, consider the following:

- Your clusters can contain a mix of Trn1, Inf1, Inf2 and other instances.
- You need a Linux application in a container that uses a machine learning framework that supports AWS Neuron.

> **Important**
>
> Applications that use other frameworks might not have improved performance on Trn1, Inf1, and Inf2 instances.

- Only one inference or inference-training task can run on each AWS Trainium or AWS Inferentia chip. For Inf1, each chip has 4 NeuronCores. For Trn1 and Inf2 each chip has 2 NeuronCores. You can run as many tasks as there are chips for each of your Trn1, Inf1, and Inf2 instances.
- When creating a service or running a standalone task, you can use instance type attributes when you configure task placement constraints. This ensures that the task is launched on the

container instance that you specify. Doing so can help you optimize overall resource utilization and ensure that tasks for inference workloads are on your Trn1, Inf1, and Inf2 instances. For more information, see How Amazon ECS places tasks on container instances.

In the following example, a task is run on an `Inf1.xlarge` instance on your `default` cluster.

```
aws ecs run-task \
     --cluster default \
     --task-definition ecs-inference-task-def \
     --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
 Inf1.xlarge"
```

- Neuron resource requirements can't be defined in a task definition. Instead, you configure a container to use specific AWS Trainium or AWS Inferentia chips available on the host container instance. Do this by using the `linuxParameters` parameter and specifying the device details. For more information, see Task definition requirements.

Use the Amazon ECS-optimized Amazon Linux 2023 (Neuron) AMI

Amazon ECS provides an Amazon ECS optimized AMI that's based on Amazon Linux 2023 for AWS Trainium and AWS Inferentia workloads. It comes with the AWS Neuron drivers and runtime for Docker. This AMI makes running machine learning inference workloads easier on Amazon ECS.

We recommend using the Amazon ECS-optimized Amazon Linux 2023 (Neuron) AMI when launching your Amazon EC2 Trn1, Inf1, and Inf2 instances.

You can retrieve the current Amazon ECS-optimized Amazon Linux 2023 (Neuron) AMI using the AWS CLI with the following command.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/neuron/
recommended
```

The Amazon ECS-optimized Amazon Linux 2023 (Neuron) AMI is supported in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)

- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Tokyo)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- South America (São Paulo)

Use the Amazon ECS optimized Amazon Linux 2 (Neuron) AMI

Amazon ECS provides an Amazon ECS optimized AMI that's based on Amazon Linux 2 for AWS Trainium and AWS Inferentia workloads. It comes with the AWS Neuron drivers and runtime for Docker. This AMI makes running machine learning inference workloads easier on Amazon ECS.

We recommend using the Amazon ECS optimized Amazon Linux 2 (Neuron) AMI when launching your Amazon EC2 Trn1, Inf1, and Inf2 instances.

You can retrieve the current Amazon ECS optimized Amazon Linux 2 (Neuron) AMI using the AWS CLI with the following command.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/inf/
recommended
```

The Amazon ECS optimized Amazon Linux 2 (Neuron) AMI is supported in the following Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)

- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Tokyo)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- South America (São Paulo)

Task definition requirements

To deploy Neuron on Amazon ECS, your task definition must contain the container definition for a pre-built container serving the inference model for TensorFlow. It's provided by AWS Deep Learning Containers. This container contains the AWS Neuron runtime and the TensorFlow Serving application. At startup, this container fetches your model from Amazon S3, launches Neuron TensorFlow Serving with the saved model, and waits for prediction requests. In the following example, the container image has TensorFlow 1.15 and Ubuntu 18.04. A complete list of pre-built Deep Learning Containers optimized for Neuron is maintained on GitHub. For more information, see Using AWS Neuron TensorFlow Serving.

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.4-neuron-py37-ubuntu18.04
```

Alternatively, you can build your own Neuron sidecar container image. For more information, see Tutorial: Neuron TensorFlow Serving in the *AWS Deep Learning AMIs Developer Guide*.

The task definition must be specific to a single instance type. You must configure a container to use specific AWS Trainium or AWS Inferentia devices that are available on the host container instance. You can do so using the `linuxParameters` parameter. The following table details the chips that are specific to each instance type.

Instance Type	vCPUs	RAM (GiB)	AWS ML accelerator chips	Device Paths
trn1.2xlarge	8	32	1	`/dev/neuron0`
trn1.32xlarge	128	512	16	`/dev/neuron0` , `/dev/neuron1` , `/dev/neuron2` , `/dev/neuron3` , `/dev/neuron4` , `/dev/neuron5` , `/dev/neuron6` , `/dev/neuron7` , `/dev/neuron8` , `/dev/neuron9` , `/dev/neuron10` , `/dev/neuron11` , `/dev/neuron12` , `/dev/neuron13` , `/dev/neuron14` , `/dev/neuron15`
inf1.xlarge	4	8	1	`/dev/neuron0`
inf1.2xlarge	8	16	1	`/dev/neuron0`

Instance Type	vCPUs	RAM (GiB)	AWS ML accelerator chips	Device Paths
inf1.6xlarge	24	48	4	/dev/neuron0 , /dev/neuron1 , /dev/neuron2 , /dev/neuron3
inf1.24xlarge	96	192	16	/dev/neuron0 , /dev/neuron1 , /dev/neuron2 , /dev/neuron3 , /dev/neuron4 , /dev/neuron5 , /dev/neuron6 , /dev/neuron7 , /dev/neuron8 , /dev/neuron9 , /dev/neuron10 , /dev/neuron11 , /dev/neuron12 , /dev/neuron13 , /dev/neuron14 , /dev/neuron15

Instance Type	vCPUs	RAM (GiB)	AWS ML accelerator chips	Device Paths
inf2.xlarge	8	16	1	/dev/neuron0
inf2.8xlarge	32	64	1	/dev/neuron0
inf2.24xlarge	96	384	6	/dev/neuron0 , /dev/neuron1 , /dev/neuron2 , /dev/neuron3 , /dev/neuron4 , /dev/neuron5 ,
inf2.48xlarge	192	768	12	/dev/neuron0 , /dev/neuron1 , /dev/neuron2 , /dev/neuron3 , /dev/neuron4 , /dev/neuron5 , /dev/neuron6 , /dev/neuron7 , /dev/neuron8 , /dev/neuron9 , /dev/neuron10 , /dev/neuron11

Specifying AWS Neuron machine learning in an Amazon ECS task definition

The following is an example Linux task definition for `inf1.xlarge`, displaying the syntax to use.

```
{
    "family": "ecs-neuron",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == inf1.xlarge"
        }
    ],
    "executionRoleArn": "${YOUR_EXECUTION_ROLE}",
    "containerDefinitions": [
        {
            "entryPoint": [
                "/usr/local/bin/entrypoint.sh",
                "--port=8500",
                "--rest_api_port=9000",
                "--model_name=resnet50_neuron",
                "--model_base_path=s3://amzn-s3-demo-bucket/resnet50_neuron/"
            ],
            "portMappings": [
                {
                    "hostPort": 8500,
                    "protocol": "tcp",
                    "containerPort": 8500
                },
                {
                    "hostPort": 8501,
                    "protocol": "tcp",
                    "containerPort": 8501
                },
                {
                    "hostPort": 0,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
```

```
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/neuron0",
                        "hostPath": "/dev/neuron0",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ],
                "capabilities": {
                    "add": [
                        "IPC_LOCK"
                    ]
                }
            },
            "cpu": 0,
            "memoryReservation": 1000,
            "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-
inference-neuron:1.15.4-neuron-py37-ubuntu18.04",
            "essential": true,
            "name": "resnet50"
        }
    ]
}
```

Amazon ECS task definitions for deep learning instances

To use deep learning workloads on Amazon ECS, register Amazon EC2 DL1 instances to your clusters. Amazon EC2 DL1 instances are powered by Gaudi accelerators from Habana Labs (an Intel company). Use the Habana SynapseAI SDK to connect to the Habana Gaudi accelerators. The SDK supports the popular machine learning frameworks, TensorFlow and PyTorch.

Considerations

Before you begin deploying DL1 on Amazon ECS, consider the following:

- Your clusters can contain a mix of DL1 and non-DL1 instances.
- When creating a service or running a standalone task, you can use instance type attributes specifically when you configure task placement constraints to ensure that your task is launched on the container instance that you specify. Doing so ensures that your resources are used

effectively and that your tasks for deep learning workloads are on your DL1 instances. For more information, see How Amazon ECS places tasks on container instances.

The following example runs a task on a `dl1.24xlarge` instance on your `default` cluster.

```
aws ecs run-task \
     --cluster default \
     --task-definition ecs-dl1-task-def \
     --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
 dl1.24xlarge"
```

Using a DL1 AMI

You have three options for running an AMI on Amazon EC2 DL1 instances for Amazon ECS:

- AWS Marketplace AMIs that are provided by Habana here.
- Habana Deep Learning AMIs that are provided by Amazon Web Services. Because it's not included, you need to install the Amazon ECS container agent separately.
- Use Packer to build a custom AMI that's provided by the GitHub repo. For more information, see the Packer documentation.

Specifying deep learning in an Amazon ECS task definition

To run Habana Gaudi accelerated deep learning containers on Amazon ECS, your task definition must contain the container definition for a pre-built container that serves the deep learning model for TensorFlow or PyTorch using Habana SynapseAI that's provided by AWS Deep Learning Containers.

The following container image has TensorFlow 2.7.0 and Ubuntu 20.04. A complete list of pre-built Deep Learning Containers that's optimized for the Habana Gaudi accelerators is maintained on GitHub. For more information, see Habana Training Containers.

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training-habana:2.7.0-hpu-py38-
synapseai1.2.0-ubuntu20.04
```

The following is an example task definition for Linux containers on Amazon EC2, displaying the syntax to use. This example uses an image containing the Habana Labs System Management

Interface Tool (HL-SMI) found here: `vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/tensorflow-installer-tf-cpu-2.6.0:1.1.0-614`

```
{
    "family": "dl-test",
    "requiresCompatibilities": ["EC2"],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.os-type == linux"
        },
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type == dl1.24xlarge"
        }
    ],
    "networkMode": "host",
    "cpu": "10240",
    "memory": "1024",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": ["hl-smi"],
            "cpu": 8192,
            "environment": [
                {
                    "name": "HABANA_VISIBLE_DEVICES",
                    "value": "all"
                }
            ],
            "image": "vault.habana.ai/gaudi-docker/1.1.0/ubuntu20.04/habanalabs/
tensorflow-installer-tf-cpu-2.6.0:1.1.0-614",
            "essential": true,
            "name": "tensorflow-installer-tf-hpu"
        }
    ]
}
```

Amazon ECS task definitions for 64-bit ARM workloads

Amazon ECS supports using 64-bit ARM applications. You can run your applications on the platform that's powered by AWS Graviton Processors. It's suitable for a wide variety of workloads. This includes workloads such as application servers, micro-services, high-performance computing, CPU-based machine learning inference, video encoding, electronic design automation, gaming, open-source databases, and in-memory caches.

Considerations

Before you begin deploying task definitions that use the 64-bit ARM architecture, consider the following:

- The applications can use the Fargate or EC2 launch types.
- The applications can only use the Linux operating system.
- For the Fargate type, the applications must use Fargate platform version `1.4.0` or later.
- The applications can use Fluent Bit or CloudWatch for monitoring.
- For the Fargate launch type, the following AWS Regions do not support 64-bit ARM workloads:
  - US East (N. Virginia), the `use1-az3` Availability Zone
- For the Amazon EC2 launch type, see the following to verify that the Region that you're in supports the instance type you want to use:
  - Amazon EC2 M6g Instances
  - Amazon EC2 T4g Instances
  - Amazon EC2 C6g Instances
  - Amazon EC2 R6gd Instances
  - Amazon EC2 X2gd Instances

  You can also use the Amazon EC2 `describe-instance-type-offerings` command with a filter to view the instance offering for your Region.

  ```
  aws ec2 describe-instance-type-offerings --filters Name=instance-type,Values=instance-type --region region
  ```

  The following example checks for the M6 instance type availability in the US East (N. Virginia) (us-east-1) Region.

```
aws ec2 describe-instance-type-offerings --filters "Name=instance-type,Values=m6*" --
region us-east-1
```

For more information, see describe-instance-type-offerings in the *Amazon EC2 Command Line Reference*.

Specifying the ARM architecture in an Amazon ECS task definition

To use the ARM architecture, specify `ARM64` for the `cpuArchitecture` task definition parameter.

In the following example, the ARM architecture is specified in a task definition. It's in JSON format.

```
{
    "runtimePlatform": {
        "operatingSystemFamily": "LINUX",
        "cpuArchitecture": "ARM64"
    },
...
}
```

In the following example, a task definition for the ARM architecture displays "hello world."

```
{
 "family": "arm64-testapp",
 "networkMode": "awsvpc",
 "containerDefinitions": [
    {
        "name": "arm-container",
        "image": "public.ecr.aws/docker/library/busybox:latest",
        "cpu": 100,
        "memory": 100,
        "essential": true,
        "command": [ "echo hello world" ],
        "entryPoint": [ "sh", "-c" ]
    }
 ],
 "requiresCompatibilities": [ "EC2" ],
 "cpu": "256",
 "memory": "512",
 "runtimePlatform": {
```

```
        "operatingSystemFamily": "LINUX",
        "cpuArchitecture": "ARM64"
  },
 "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole"
}
```

Send Amazon ECS logs to CloudWatch

You can configure the containers in your tasks to send log information to CloudWatch Logs. If you're using the Fargate launch type for your tasks, you can view the logs from your containers. If you're using the EC2 launch type, you can view different logs from your containers in one convenient location, and it prevents your container logs from taking up disk space on your container instances.

> **Note**
>
> The type of information that is logged by the containers in your task depends mostly on their `ENTRYPOINT` command. By default, the logs that are captured show the command output that you typically might see in an interactive terminal if you ran the container locally, which are the `STDOUT` and `STDERR` I/O streams. The `awslogs` log driver simply passes these logs from Docker to CloudWatch Logs. For more information about how Docker logs are processed, including alternative ways to capture different file data or streams, see View logs for a container or service in the Docker documentation.

To send system logs from your Amazon ECS container instances to CloudWatch Logs, see Monitoring Log Files and CloudWatch Logs quotas in the *Amazon CloudWatch Logs User Guide*.

Fargate launch type

If you're using the Fargate launch type for your tasks, you need to add the required `logConfiguration` parameters to your task definition to turn on the `awslogs` log driver. For more information, see Example Amazon ECS task definition: Route logs to CloudWatch.

For Windows container on Fargate, perform one of the following options when any of your task definition parameters have special characters such as, `& \ < > ^ |`:

- Add an escape (\) with double quotes around the entire parameter string

  Example

```
"awslogs-multiline-pattern": "\"^[|DEBUG|INFO|WARNING|ERROR\"",
```

- Add an escape (^) character around each special character

  Example

```
"awslogs-multiline-pattern": "^^[^|DEBUG^|INFO^|WARNING^|ERROR",
```

EC2 launch type

If you're using the EC2 launch type for your tasks and want to turn on the `awslogs` log driver, your Amazon ECS container instances require at least version 1.9.0 of the container agent. For information about how to check your agent version and updating to the latest version, see Updating the Amazon ECS container agent.

> **Note**
>
> You must use either an Amazon ECS-optimized AMI or a custom AMI with at least version `1.9.0-1` of the `ecs-init` package. When using a custom AMI, you must specify that the `awslogs` logging driver is available on the Amazon EC2 instance when you start the agent by using the following environment variable in your **docker run** statement or environment variable file.
>
> ```
> ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
> ```

Your Amazon ECS container instances also require `logs:CreateLogStream` and `logs:PutLogEvents` permission on the IAM role that you can launch your container instances with. If you created your Amazon ECS container instance role before `awslogs` log driver support was enabled in Amazon ECS, you might need to add this permission. The `ecsTaskExecutionRole` is used when it's assigned to the task and likely contains the correct permissions. For information about the task execution role, see Amazon ECS task execution IAM role. If your container instances use the managed IAM policy for container instances, your container instances likely have the correct permissions. For information about the managed IAM policy for container instances, see Amazon ECS container instance IAM role.

Example Amazon ECS task definition: Route logs to CloudWatch

Before your containers can send logs to CloudWatch, you must specify the `awslogs` log driver for containers in your task definition. For more information about the log parameters, see Storage and logging

The task definition JSON that follows has a `logConfiguration` object specified for each container. One is for the WordPress container that sends logs to a log group called `awslogs-wordpress`. The other is for a MySQL container that sends logs to a log group that's called `awslogs-mysql`. Both containers use the `awslogs-example` log stream prefix.

```
{
    "containerDefinitions": [
        {
            "name": "wordpress",
            "links": [
                "mysql"
            ],
            "image": "public.ecr.aws/docker/library/wordpress:latest",
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-wordpress",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            },
            "memory": 500,
            "cpu": 10
        },
        {
            "environment": [
                {
                    "name": "MYSQL_ROOT_PASSWORD",
                    "value": "password"
```

```
                }
            ],
            "name": "mysql",
            "image": "public.ecr.aws/docker/library/mysql:latest",
            "cpu": 10,
            "memory": 500,
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "awslogs-mysql",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example",
                    "mode": "non-blocking",
                    "max-buffer-size": "25m"
                }
            }
        }
    ],
    "family": "awslogs-example"
}
```

Next steps

- You can optionally set a retention policy for the log group by using the CloudWatch AWS CLI or API. For more information, see put-retention-policy in the *AWS Command Line Interface Reference*.
- After you have registered a task definition with the `awslogs` log driver in a container definition log configuration, you can run a task or create a service with that task definition to start sending logs to CloudWatch Logs. For more information, see Running an application as an Amazon ECS task and Creating an Amazon ECS rolling update deployment.

Send Amazon ECS logs to an AWS service or AWS Partner

You can use FireLens for Amazon ECS to use task definition parameters to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics. The AWS Partner Network is a global community of partners that leverages programs, expertise, and resources to build, market, and sell customer offerings. For more information see AWS Partner. FireLens works with Fluentd and Fluent Bit. We provide the AWS for Fluent Bit image or you can use your own Fluentd or Fluent Bit image.

By default, Amazon ECS configures the container dependency so that the Firelens container starts before any container that uses it. The Firelens container also stops after all containers that use it stop.

Consider the following when using FireLens for Amazon ECS:

- We recommend that you add `my_service_` to the log container name so that you can easily distinguish container names in the console.
- Amazon ECS adds a start container order dependency between the application containers and the FireLens container by default. When you specify a container order between the application containers and the FireLens container, then the default start container order is overridden.
- FireLens for Amazon ECS is supported for tasks that are hosted on both AWS Fargate on Linux and Amazon EC2 on Linux. Windows containers don't support FireLens.

  For information about how to configure centralized logging for Windows containers, see Centralized logging for Windows containers on Amazon ECS using Fluent Bit.
- You can use AWS CloudFormation templates to configure FireLens for Amazon ECS. For more information, see AWS::ECS::TaskDefinition FirelensConfiguration in the *AWS CloudFormation User Guide*
- FireLens listens on port `24224`, so to ensure that the FireLens log router isn't reachable outside of the task, you must not allow inbound traffic on port `24224` in the security group your task uses. For tasks that use the `awsvpc` network mode, this is the security group associated with the task. For tasks using the `host` network mode, this is the security group that's associated with the Amazon EC2 instance hosting the task. For tasks that use the `bridge` network mode, don't create any port mappings that use port `24224`.
- For tasks that use the `bridge` network mode, the container with the FireLens configuration must start before any application containers that rely on it start. To control the start order of your containers, use dependency conditions in your task definition. For more information, see Container dependency.

> **Note**
>
> If you use dependency condition parameters in container definitions with a FireLens configuration, ensure that each container has a START or HEALTHY condition requirement.

- By default, FireLens adds the cluster and task definition name and the Amazon Resource Name (ARN) of the cluster as metadata keys to your stdout/stderr container logs. The following is an example of the metadata format.

```
"ecs_cluster": "cluster-name",
"ecs_task_arn": "arn:aws:ecs:region:111122223333:task/cluster-
name/f2ad7dba413f45ddb4EXAMPLE",
"ecs_task_definition": "task-def-name:revision",
```

  If you do not want the metadata in your logs, set `enable-ecs-log-metadata` to `false` in the `firelensConfiguration` section of the task definition.

```
"firelensConfiguration":{
   "type":"fluentbit",
   "options":{
      "enable-ecs-log-metadata":"false",
      "config-file-type":"file",
      "config-file-value":"/extra.conf"
}
```

To use this feature, you must create an IAM role for your tasks that provides the permissions necessary to use any AWS services that the tasks require. For example, if a container is routing logs to Firehose, the task requires permission to call the `firehose:PutRecordBatch` API. For more information, see Adding and Removing IAM Identity Permissions in the *IAM User Guide*.

Your task might also require the Amazon ECS task execution role under the following conditions. For more information, see Amazon ECS task execution IAM role.

- If your task is hosted on Fargate and you are pulling container images from Amazon ECR or referencing sensitive data from AWS Secrets Manager in your log configuration, then you must include the task execution IAM role.

- When you use a custom configuration file that's hosted in Amazon S3, your task execution IAM role must include the `s3:GetObject` permission.

For information about how to use multiple configuration files with Amazon ECS, including files that you host or files in Amazon S3, see Init process for Fluent Bit on ECS, multi-config support.

For information about example configurations, see Example Amazon ECS task definition: Route logs to FireLens

Configuring Amazon ECS logs for high throughput

When you create a task definition, you can specify the number of log lines that are buffered in memory by specifying the value in the `log-driver-buffer-limit`. For more information, see Fluentd logging driver in the Docker documentation.

Use this option when there's high throughput, because Docker might run out of buffer memory and discard buffer messages, so it can add new messages.

Consider the following when using FireLens for Amazon ECS with the buffer limit option:

- This option is supported on the Amazon EC2 launch type and the Fargate launch type with platform version `1.4.0` or later.
- The option is only valid when `logDriver` is set to `awsfirelens`.
- The default buffer limit is `1048576` log lines.
- The buffer limit must be greater than or equal to `0` and less than `536870912` log lines.
- The maximum amount of memory used for this buffer is the product of the size of each log line and the size of the buffer. For example, if the application's log lines are on average 2 KiB, a buffer limit of 4096 would use at most 8 MiB. The total amount of memory allocated at the task level should be greater than the amount of memory that's allocated for all the containers in addition to the log driver memory buffer.

When the `awsfirelens` log driver is specified in a task definition, the Amazon ECS container agent injects the following environment variables into the container:

`FLUENT_HOST`

The IP address that's assigned to the FireLens container.

> **Note**
>
> If you're using the EC2 launch type with the `bridge` network mode, the `FLUENT_HOST` environment variable in your application container can become inaccurate after a restart of the FireLens log router container (the container with the `firelensConfiguration` object in its container definition). This is because `FLUENT_HOST` is a dynamic IP address and can change after a restart. Logging directly from the application container to the `FLUENT_HOST` IP address can start failing after the address changes. For more information about restarting individual containers, see [Restart individual containers in Amazon ECS tasks with container restart policies](#).

`FLUENT_PORT`

The port that the Fluent Forward protocol is listening on.

You can use the `FLUENT_HOST` and `FLUENT_PORT` environment variables to log directly to the log router from code instead of going through `stdout`. For more information, see [fluent-logger-golang](#) on GitHub.

The following shows the syntax for specifying the `log-driver-buffer-limit`. Replace `my_service_` with the name of your service:

```
{
    "containerDefinitions": [
        {
            "name": "my_service_log_router",
            "image": "public.ecr.aws/aws-observability/aws-for-fluent-bit:stable",
            "cpu": 0,
            "memoryReservation": 51,
            "portMappings": [],
            "essential": true,
            "environment": [],
            "mountPoints": [],
            "volumesFrom": [],
            "user": "0",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/ecs-aws-firelens-sidecar-container",
```

```
                    "mode": "non-blocking",
                    "awslogs-create-group": "true",
                    "max-buffer-size": "25m",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "firelens"
                },
                "secretOptions": []
            },
            "systemControls": [],
            "firelensConfiguration": {
                "type": "fluentbit"
            }
        },
        {
            "essential": true,
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "firehose",
                    "region": "us-west-2",
                    "delivery_stream": "my-stream",
                    "log-driver-buffer-limit": "51200"
                }
            },
            "dependsOn": [
                {
                    "containerName": "log_router",
                    "condition": "START"
                }
            ],
            "memoryReservation": 100
        }
    ]
}
```

AWS for Fluent Bit image repositories for Amazon ECS

AWS provides a Fluent Bit image with plugins for both CloudWatch Logs and Firehose. We recommend using Fluent Bit as your log router because it has a lower resource utilization rate than Fluentd. For more information, see CloudWatch Logs for Fluent Bit and Amazon Kinesis Firehose for Fluent Bit.

The **AWS for Fluent Bit** image is available on Amazon ECR on both the Amazon ECR Public Gallery and in an Amazon ECR repository in most AWS Regions for high availability.

Amazon ECR Public Gallery

The AWS for Fluent Bit image is available on the Amazon ECR Public Gallery. This is the recommended location to download the AWS for Fluent Bit image because it's a public repository and available to be used from all AWS Regions. For more information, see aws-for-fluent-bit on the Amazon ECR Public Gallery.

Linux

The AWS for Fluent Bit image in the Amazon ECR Public Gallery supports Amazon Linux operating system with the `ARM 64`, or `x86-64` architecture.

You can pull the AWS for Fluent Bit image from the Amazon ECR Public Gallery by specifying the repository URL with the desired image tag. The available image tags can be found on the **Image tags** tab on the Amazon ECR Public Gallery.

The following shows the syntax to use for the Docker CLI.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

For example, you can pull the latest stable AWS for Fluent Bit image using this Docker CLI command.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:stable
```

> **Note**
>
> Unauthenticated pulls are allowed, but have a lower rate limit than authenticated pulls. To authenticate using your AWS account before pulling, use the following command.
>
> ```
> aws ecr-public get-login-password --region us-east-1 | docker login --username
>  AWS --password-stdin public.ecr.aws
> ```

Windows

The AWS for Fluent Bit image in the Amazon ECR Public Gallery supports the AMD64 architecture with the following operating systems:

- Windows Server 2022 Full
- Windows Server 2022 Core
- Windows Server 2019 Full
- Windows Server 2019 Core

Windows containers that are on AWS Fargate don't support FireLens.

You can pull the AWS for Fluent Bit image from the Amazon ECR Public Gallery by specifying the repository URL with the desired image tag. The available image tags can be found on the **Image tags** tab on the Amazon ECR Public Gallery.

The following shows the syntax to use for the Docker CLI.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

For example, you can pull the newest stable AWS for Fluent Bit image using this Docker CLI command.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:windowsservercore-stable
```

> **Note**
>
> Unauthenticated pulls are allowed, but have a lower rate limit than authenticated pulls. To authenticate using your AWS account before pulling, use the following command.
>
> ```
> aws ecr-public get-login-password --region us-east-1 | docker login --username
>  AWS --password-stdin public.ecr.aws
> ```

Amazon ECR

The AWS for Fluent Bit image is available on Amazon ECR for high availability. These images are available in most AWS Regions, including AWS GovCloud (US).

Linux

The latest stable AWS for Fluent Bit image URI can be retrieved using the following command.

```
aws ssm get-parameters \
      --names /aws/service/aws-for-fluent-bit/stable \
      --region us-east-1
```

All versions of the AWS for Fluent Bit image can be listed using the following command to query the Systems Manager Parameter Store parameter.

```
aws ssm get-parameters-by-path \
      --path /aws/service/aws-for-fluent-bit \
      --region us-east-1
```

The newest stable AWS for Fluent Bit image can be referenced in an AWS CloudFormation template by referencing the Systems Manager parameter store name. The following is an example:

```
Parameters:
  FireLensImage:
    Description: Fluent Bit image for the FireLens Container
    Type: AWS::SSM::Parameter::Value<String>
    Default: /aws/service/aws-for-fluent-bit/stable
```

Windows

The latest stable AWS for Fluent Bit image URI can be retrieved using the following command.

```
aws ssm get-parameters \
      --names /aws/service/aws-for-fluent-bit/windowsservercore-stable \
      --region us-east-1
```

All versions of the AWS for Fluent Bit image can be listed using the following command to query the Systems Manager Parameter Store parameter.

```
aws ssm get-parameters-by-path \
```

```
    --path /aws/service/aws-for-fluent-bit/windowsservercore \
    --region us-east-1
```

The latest stable AWS for Fluent Bit image can be referenced in an AWS CloudFormation template by referencing the Systems Manager parameter store name. The following is an example:

```
Parameters:
  FireLensImage:
    Description: Fluent Bit image for the FireLens Container
    Type: AWS::SSM::Parameter::Value<String>
    Default: /aws/service/aws-for-fluent-bit/windowsservercore-stable
```

Example Amazon ECS task definition: Route logs to FireLens

To use custom log routing with FireLens, you must specify the following in your task definition:

- A log router container that contains a FireLens configuration. We recommend that the container be marked as `essential`.
- One or more application containers that contain a log configuration specifying the `awsfirelens` log driver.
- A task IAM role Amazon Resource Name (ARN) that contains the permissions needed for the task to route the logs.

When creating a new task definition using the AWS Management Console, there is a FireLens integration section that makes it easy to add a log router container. For more information, see [Creating an Amazon ECS task definition using the console](#).

Amazon ECS converts the log configuration and generates the Fluentd or Fluent Bit output configuration. The output configuration is mounted in the log routing container at `/fluent-bit/etc/fluent-bit.conf` for Fluent Bit and `/fluentd/etc/fluent.conf` for Fluentd.

> **Important**
>
> FireLens listens on port 24224. Therefore, to ensure that the FireLens log router isn't reachable outside of the task, you must not allow ingress traffic on port 24224 in the security group your task uses. For tasks that use the `awsvpc` network mode, this is the security group that's associated with the task. For tasks that use the `host` network mode, this is the security group that's associated with the Amazon EC2 instance hosting the task.

For tasks that use the `bridge` network mode, don't create any port mappings that use port `24224`.

By default, Amazon ECS adds additional fields in your log entries that help identify the source of the logs.

- `ecs_cluster` – The name of the cluster that the task is part of.
- `ecs_task_arn` – The full Amazon Resource Name (ARN) of the task that the container is part of.
- `ecs_task_definition` – The task definition name and revision that the task is using.
- `ec2_instance_id` – The Amazon EC2 instance ID that the container is hosted on. This field is only valid for tasks using the EC2 launch type.

You can set the `enable-ecs-log-metadata` to `false` if you do not want the metadata.

The following task definition example defines a log router container that uses Fluent Bit to route its logs to CloudWatch Logs. It also defines an application container that uses a log configuration to route logs to Amazon Data Firehose and sets the memory that's used to buffer events to the 2 MiB.

Note

For more example task definitions, see [Amazon ECS FireLens examples](#) on GitHub.

```
{
  "family": "firelens-example-firehose",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
  "containerDefinitions": [
    {
            "name": "log_router",
            "image": "public.ecr.aws/aws-observability/aws-for-fluent-bit:stable",
            "cpu": 0,
            "memoryReservation": 51,
            "portMappings": [],
            "essential": true,
            "environment": [],
            "mountPoints": [],
            "volumesFrom": [],
```

```
            "user": "0",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/ecs-aws-firelens-sidecar-container",
                    "mode": "non-blocking",
                    "awslogs-create-group": "true",
                    "max-buffer-size": "25m",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "firelens"
                },
                "secretOptions": []
            },
            "systemControls": [],
            "firelensConfiguration": {
                "type": "fluentbit"
            }
        },
    {
      "essential": true,
      "image": "public.ecr.aws/docker/library/httpd:latest",
      "name": "app",
      "logConfiguration": {
        "logDriver": "awsfirelens",
        "options": {
          "Name": "firehose",
          "region": "us-west-2",
          "delivery_stream": "my-stream",
          "log-driver-buffer-limit": "2097152"
        }
      },
      "memoryReservation": 100
    }
  ]
}
```

The key-value pairs specified as options in the `logConfiguration` object are used to generate the Fluentd or Fluent Bit output configuration. The following is a code example from a Fluent Bit output definition.

```
[OUTPUT]
    Name   firehose
    Match  app-firelens*
```

```
    region us-west-2
    delivery_stream my-stream
```

> **Note**
>
> FireLens manages the `match` configuration. You do not specify the `match` configuration in your task definition.

Use a custom configuration file

You can specify a custom configuration file. The configuration file format is the native format for the log router that you're using. For more information, see Fluentd Config File Syntax and YAML Configuration.

In your custom configuration file, for tasks using the `bridge` or `awsvpc` network mode, don't set a Fluentd or Fluent Bit forward input over TCP because FireLens adds it to the input configuration.

Your FireLens configuration must contain the following options to specify a custom configuration file:

`config-file-type`

The source location of the custom configuration file. The available options are `s3` or `file`.

> **Note**
>
> Tasks that are hosted on AWS Fargate only support the `file` configuration file type.

`config-file-value`

The source for the custom configuration file. If the `s3` config file type is used, the config file value is the full ARN of the Amazon S3 bucket and file. If the `file` config file type is used, the config file value is the full path of the configuration file that exists either in the container image or on a volume that's mounted in the container.

> **Important**
>
> When using a custom configuration file, you must specify a different path than the one FireLens uses. Amazon ECS reserves the `/fluent-bit/etc/fluent-bit.conf` filepath for Fluent Bit and `/fluentd/etc/fluent.conf` for Fluentd.

The following example shows the syntax required when specifying a custom configuration.

> **Important**
>
> To specify a custom configuration file that's hosted in Amazon S3, ensure you have created a task execution IAM role with the proper permissions.

The following shows the syntax required when specifying a custom configuration.

```
{
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-
bit:stable",
      "name": "log_router",
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "config-file-type": "s3 | file",
          "config-file-value": "arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf
 | filepath"
        }
      }
    }
  ]
}
```

> **Note**
>
> Tasks hosted on AWS Fargate only support the `file` configuration file type.

Using non-AWS container images in Amazon ECS

Use private registry to store your credentials in AWS Secrets Manager, and then reference them in your task definition. This provides a way to reference container images that exist in private registries outside of AWS that require authentication in your task definitions. This feature is supported by tasks hosted on Fargate, Amazon EC2 instances, and external instances using Amazon ECS Anywhere.

> **Important**
>
> If your task definition references an image that's stored in Amazon ECR, this topic doesn't apply. For more information, see Using Amazon ECR Images with Amazon ECS in the *Amazon Elastic Container Registry User Guide*.

For tasks hosted on Amazon EC2 instances, this feature requires version `1.19.0` or later of the container agent. However, we recommend using the latest container agent version. For information about how to check your agent version and update to the latest version, see Updating the Amazon ECS container agent.

For tasks hosted on Fargate, this feature requires platform version `1.2.0` or later. For information, see Fargate platform versions for Amazon ECS.

Within your container definition, specify the `repositoryCredentials` object with the details of the secret that you created. The referenced secret can be from a different AWS Region or a different account than the task using it.

> **Note**
>
> When using the Amazon ECS API, AWS CLI, or AWS SDK, if the secret exists in the same AWS Region as the task that you're launching then you can use either the full ARN or name of the secret. If the secret exists in a different account, the full ARN of the secret must be specified. When using the AWS Management Console, the full ARN of the secret must be specified always.

The following is a snippet of a task definition that shows the required parameters:

Substitute the following parameters:

- *private-repo* with the private repository host name
- *private-image* with the image name
- *arn:aws:secretsmanager:region:aws_account_id:secret:secret_name* with the secret Amazon Resource Name (ARN)

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

> **Note**
>
> Another method of enabling private registry authentication uses Amazon ECS container agent environment variables to authenticate to private registries. This method is only supported for tasks hosted on Amazon EC2 instances. For more information, see Configuring Amazon ECS container instances for private Docker images .

To use private registry

1. The task definition must have a task execution role. This allows the container agent to pull the container image. For more information, see Amazon ECS task execution IAM role.

   Private registry authentication allows your Amazon ECS tasks to pull container images from private registries outside of AWS (such as Docker Hub, Quay.io, or your own private registry) that require authentication credentials. This feature uses Secrets Manager to securely store your registry credentials, which are then referenced in your task definition using the `repositoryCredentials` parameter.

   For more information about configuring private registry authentication, see Using non-AWS container images in Amazon ECS.

To provide access to the secrets that contain your private registry credentials, add the following permissions as an inline policy to the task execution role. For more information, see Adding and Removing IAM Policies.

- `secretsmanager:GetSecretValue`—Required to retrieve the private registry credentials from Secrets Manager.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The Amazon Resource Name (ARN) for your custom key must be added as a resource.

The following is an example inline policy that adds the permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [

 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:secret_name",
                "arn:aws:kms:<region>:<aws_account_id>:key/key_id"
            ]
        }
    ]
}
```

2. Use AWS Secrets Manager to create a secret for your private registry credentials. For information about how to create a secret, see Create an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

   Enter your private registry credentials using the following format:

```
{
  "username" : "privateRegistryUsername",
  "password" : "privateRegistryPassword"
}
```

3. Register a task definition. For more information, see the section called “Creating a task definition using the console”.

Restart individual containers in Amazon ECS tasks with container restart policies

You can enable a restart policy for each essential and non-essential container defined in your task definition, to overcome transient failures faster and maintain task availability. When you enable a restart policy for a container, Amazon ECS can restart the container if it exits, without needing to replace the task.

Restart policies are not enabled for containers by default. When you enable a restart policy for a container, you can specify exit codes that the container will not be restarted on. These can be exit codes that indicate success, like exit code `0`, that don't require a restart. You can also specify how long a container must run succesfully before a restart can be attempted. For more information about these parameters, see Restart policy. For an example task definition that specifies these values, see Specifying a container restart policy in an Amazon ECS task definition.

You can use the Amazon ECS task metadata endpoint or CloudWatch Container Insights to monitor the number of times a container has restarted. For more information about the task metadata endpoint, see Amazon ECS task metadata endpoint version 4 and Amazon ECS task metadata endpoint version 4 for tasks on Fargate. For more information about Container Insights metrics for Amazon ECS, see Amazon ECS Container Insights metrics in the *Amazon CloudWatch User Guide*.

Container restart policies are supported by tasks hosted on Fargate, Amazon EC2 instances, and external instances using Amazon ECS Anywhere.

Considerations

Consider the following before enabling a restart policy for your container:

- Restart policies aren't supported for Windows containers on Fargate.
- For tasks hosted on Amazon EC2 instances, this feature requires version `1.86.0` or later of the container agent. However, we recommend using the latest container agent version. For information about how to check your agent version and update to the latest version, see Updating the Amazon ECS container agent.
- For tasks hosted on Fargate, this feature requires platform version `1.4.0` or later. For information, see Fargate platform versions for Amazon ECS.

- If you're using the EC2 launch type with the `bridge` network mode, the `FLUENT_HOST` environment variable in your application container can become inaccurate after a restart of the FireLens log router container (the container with the `firelensConfiguration` object in its container definition). This is because `FLUENT_HOST` is a dynamic IP address and can change after a restart. Logging directly from the application container to the `FLUENT_HOST` IP address can start failing after the address changes. For more information about `FLUENT_HOST`, see Configuring Amazon ECS logs for high throughput.
- The Amazon ECS agent handles the container restart policies. If for some unexpected reason the Amazon ECS agent fails or is no longer running, the container won't be restarted.
- The restart attempt period defined in your policy determines the period of time (in seconds) that the container must run for before Amazon ECS restarts a container.

Specifying a container restart policy in an Amazon ECS task definition

To specify a restart policy for a container in a task definition, within the container definition, specify the `restartPolicy` object. For more information about the `restartPolicy` object, see Restart policy.

The following is a task definition using the Linux containers on Fargate launch type that sets up a web server. The container definition includes the `restartPolicy` object, with `enabled` set to true to enable a restart policy for the container. The container must run for 180 seconds before it can be restarted and will not be restarted if it exits with the exit code `0`, which indicates success.

```
{
  "containerDefinitions": [
    {
      "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
 <style>body {margin-top: 40px; background-color: #333;} </style> </head><body>
 <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
      ],
      "entryPoint": ["sh", "-c"],
      "essential": true,
      "image": "public.ecr.aws/docker/library/httpd:2.4",
      "logConfiguration": {
        "logDriver": "awslogs",
```

```
        "options": {
          "awslogs-group": "/ecs/fargate-task-definition",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "name": "sample-fargate-app",
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ],
      "restartPolicy": {
        "enabled": true,
        "ignoredExitCodes": [0],
        "restartAttemptPeriod": 180
      }
    }
  ],
  "cpu": "256",
  "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
  "family": "fargate-task-definition",
  "memory": "512",
  "networkMode": "awsvpc",
  "runtimePlatform": {
    "operatingSystemFamily": "LINUX"
  },
  "requiresCompatibilities": ["FARGATE"]
}
```

After you have registered a task definition with the `restartPolicy` object in a container definition, you can run a task or create a service with that task definition. For more information, see Running an application as an Amazon ECS task and Creating an Amazon ECS rolling update deployment.

Pass sensitive data to an Amazon ECS container

You can safely pass sensitive data, such as credentials to a database, into your container.

Secrets, such as API keys and database credentials, are frequently used by applications to gain access other systems. They often consist of a username and password, a certificate, or API key.

Access to these secrets should be restricted to specific IAM principals that are using IAM and injected into containers at runtime.

Secrets can be seamlessly injected into containers from AWS Secrets Manager and Amazon EC2 Systems Manager Parameter Store. These secrets can be referenced in your task as any of the following.

1. They're referenced as environment variables that use the `secrets` container definition parameter.
2. They're referenced as `secretOptions` if your logging platform requires authentication. For more information, see logging configuration options.
3. They're referenced as secrets pulled by images that use the `repositoryCredentials` container definition parameter if the registry where the container is being pulled from requires authentication. Use this method when pulling images from Amazon ECR Public Gallery. For more information, see Private registry authentication for tasks.

We recommend that you do the following when setting up secrets management.

Use AWS Secrets Manager or AWS Systems Manager Parameter Store for storing secret materials

You should securely store API keys, database credentials, and other secret materials in Secrets Manager or as an encrypted parameter in Systems Manager Parameter Store. These services are similar because they're both managed key-value stores that use AWS KMS to encrypt sensitive data. Secrets Manager, however, also includes the ability to automatically rotate secrets, generate random secrets, and share secrets across accounts. To utilize these features, use Secrets Manager. Otherwise, use encrypted parameters in Systems Manager Parameter Store.

> **Important**
>
> If your secret changes, you must force a new deployment or launch a new task to retrieve the latest secret value. For more information, see the following topics:
>
> - Tasks - Stop the task, and then start it. For more information, see Stopping an Amazon ECS task and Running an application as an Amazon ECS task.
> - Service - Update the service and use the force new deployment option. For more information, see Updating an Amazon ECS service.

Retrieve data from an encrypted Amazon S3 bucket

You should store secrets in an encrypted Amazon S3 bucket and use task roles to restrict access to those secrets. This prevents the values of environment variables from inadvertently leaking in logs and getting revealed when running `docker inspect`. When you do this, your application must be written to read the secret from the Amazon S3 bucket. For instructions, see Setting default server-side encryption behavior for Amazon S3 buckets.

Mount the secret to a volume using a sidecar container

Because there's an elevated risk of data leakage with environment variables, you should run a sidecar container that reads your secrets from AWS Secrets Manager and write them to a shared volume. This container can run and exit before the application container by using Amazon ECS container ordering. When you do this, the application container subsequently mounts the volume where the secret was written. Like the Amazon S3 bucket method, your application must be written to read the secret from the shared volume. Because the volume is scoped to the task, the volume is automatically deleted after the task stops. For an example, see the task-def.json project.

On Amazon EC2, the volume that the secret is written to can be encrypted with a AWS KMS customer managed key. On AWS Fargate, volume storage is automatically encrypted using a service managed key.

Pass an individual environment variable to an Amazon ECS container

> **Important**
>
> We recommend storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters. For more information, see Pass sensitive data to an Amazon ECS container.
> Environment variables specified in the task definition are readable by all users and roles that are allowed the `DescribeTaskDefinition` action for the task definition.

You can pass environment variables to your containers in the following ways:

- Individually using the `environment` container definition parameter. This maps to the `--env` option to **docker container run**.

- In bulk, using the `environmentFiles` container definition parameter to list one or more files that contain the environment variables. The file must be hosted in Amazon S3. This maps to the `--env-file` option to **docker run**.

The following is a snippet of a task definition showing how to specify individual environment variables.

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environment": [
                {
                    "name": "variable",
                    "value": "value"
                }
            ],
            ...
        }
    ],
    ...
}
```

Pass environment variables to an Amazon ECS container

Important

We recommend storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters. For more information, see Pass sensitive data to an Amazon ECS container.

Environment variable files are objects in Amazon S3 and all Amazon S3 security considerations apply.

You can't use the `environmentFiles` parameter on Windows containers and Windows containers on Fargate.

You can create an environment variable file and store it in Amazon S3 to pass environment variables to your container.

By specifying environment variables in a file, you can bulk inject environment variables. Within your container definition, specify the `environmentFiles` object with a list of Amazon S3 buckets containing your environment variable files.

Amazon ECS doesn't enforce a size limit on the environment variables, but a large environment variables file might fill up the disk space. Each task that uses an environment variables file causes a copy of the file to be downloaded to disk. Amazon ECS removes the file as part of the task cleanup.

For information about the supported environment variables, see Advanced container definition parameters- Environment.

Consider the following when specifying an environment variable file in a container definition.

- For Amazon ECS tasks on Amazon EC2, your container instances require that the container agent is version `1.39.0` or later to use this feature. For information about how to check your agent version and update to the latest version, see Updating the Amazon ECS container agent.
- For Amazon ECS tasks on AWS Fargate, your tasks must use platform version `1.4.0` or later (Linux) to use this feature. For more information, see Fargate platform versions for Amazon ECS.

  Verify that the variable is supported for the operating system platform. For more information, see the section called "Container definitions" and the section called "Other task definition parameters".
- The file must use the `.env` file extension and UTF-8 encoding.
- The task execution role is required to use this feature with the additional permissions for Amazon S3. This allows the container agent to pull the environment variable file from Amazon S3. For more information, see Amazon ECS task execution IAM role.
- There is a limit of 10 files per task definition.
- Each line in an environment file must contain an environment variable in `VARIABLE=VALUE` format. Spaces or quotation marks **are** included as part of the values for Amazon ECS files. Lines beginning with # are treated as comments and are ignored. For more information about the environment variable file syntax, see Set environment variables (-e, --env, --env-file) in the Docker documentation.

  The following is the appropriate syntax.

```
#This is a comment and will be ignored
VARIABLE=VALUE
ENVIRONMENT=PRODUCTION
```

- If there are environment variables specified using the `environment` parameter in a container definition, they take precedence over the variables contained within an environment file.
- If multiple environment files are specified and they contain the same variable, they're processed in order of entry. This means that the first value of the variable is used and subsequent values of duplicate variables are ignored. We recommend that you use unique variable names.
- If an environment file is specified as a container override, it's used. Moreover, any other environment files that are specified in the container definition is ignored.
- The following rules apply to the Fargate launch type:
  - The file is handled similar to a native Docker env-file.
  - Container definitions that reference environment variables that are blank and stored in Amazon S3 do not appear in the container.
  - There is no support for shell escape handling.
  - The container entry point interperts the `VARIABLE` values.

Example

The following is a snippet of a task definition showing how to specify an environment variable file.

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environmentFiles": [
                {
                    "value": "arn:aws:s3:::amzn-s3-demo-
bucket/envfile_object_name.env",
                    "type": "s3"
                }
            ],
            ...
        }
```

```
    ],
    ...
}
```

Pass Secrets Manager secrets programmatically in Amazon ECS

Instead of hardcoding sensitive information in plain text in your application, you can use Secrets Manager to store the sensitive data.

We recommend this method of retrieving sensitive data because if the Secrets Manager secret is subsequently updated, the application automatically retrieves the latest version of the secret.

Create a secret in Secrets Manager. After you create a Secrets Manager secret, update your application code to retrieve the secret.

Review the following considerations before securing sensitive data in Secrets Manager.

- Only secrets that store text data, which are secrets created with the `SecretString` parameter of the CreateSecret API, are supported. Secrets that store binary data, which are secrets created with the `SecretBinary` parameter of the CreateSecret API are not supported.
- Use interface VPC endpoints to enhance security controls. You must create the interface VPC endpoints for Secrets Manager. For information about the VPC endpoint, see Create VPC endpoints in the *AWS Secrets Manager User Guide*.
- The VPC your task uses must use DNS resolution.
- Your task definition must use a task role with the additional permissions for Secrets Manager. For more information, see Amazon ECS task IAM role.

Create the Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. For information about how to create secrets, see Create an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

Update your application to programmatically retrieve Secrets Manager secrets

You can retrieve secrets with a call to the Secrets Manager APIs directly from your application. For information, see Retrieve secrets from AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

To retrieve the sensitive data stored in the AWS Secrets Manager, see Code examples for AWS Secrets Manager using AWS SDKs in the *AWS SDK Code Examples Code Library*.

Pass Systems Manager Parameter Store secrets programmatically in Amazon ECS

Systems Manager Parameter Store provides secure storage and management of secrets. You can store data such as passwords, database strings, EC2 instance IDs and AMI IDs, and license codes as parameter values, instead of hardcoding this information in your application. You can store values as plain text or encrypted data.

We recommend this method of retrieving sensitive data because if the Systems Manager Parameter Store parameter is subsequently updated, the application automatically retrieves the latest version.

Review the following considerations before securing sensitive data in Systems Manager Parameter Store.

- Only secrets that store text data are supported. Secrets that store binary data are not supported.
- Use interface VPC endpoints to enhance security controls.
- The VPC your task uses must use DNS resolution.
- For tasks that use the EC2 launch type, you must use the Amazon ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `/etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see Amazon ECS container agent configuration.
- Your task definition must use a task role with the additional permissions for Systems Manager Parameter Store. For more information, see Amazon ECS task IAM role.

Create the parameter

You can use the Systems Manager console to create a Systems Manager Parameter Store parameter for your sensitive data. For more information, see Create a Systems Manager parameter (console) or Create a Systems Manager parameter (AWS CLI) in the *AWS Systems Manager User Guide*.

Update your application to programmatically retrieve Systems Manager Parameter Store secrets

To retrieve the sensitive data stored in the Systems Manager Parameter Store parameter, see Code examples for Systems Manager using AWS SDKs in the *AWS SDK Code Examples Code Library*.

Pass Secrets Manager secrets through Amazon ECS environment variables

When you inject a secret as an environment variable, you can specify the full contents of a secret, a specific JSON key within a secret. This helps you control the sensitive data exposed to your container. For more information about secret versioning, see What's in a Secrets Manager secret? in the *AWS Secrets Manager User Guide*.

The following should be considered when using an environment variable to inject a Secrets Manager secret into a container.

- Sensitive data is injected into your container when the container is initially started. If the secret is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- Applications that run on the container and container logs and debugging tools have access to the environment variables.
- For Amazon ECS tasks on AWS Fargate, consider the following:
  - To inject the full content of a secret as an environment variable or in a log configuration, you must use platform version `1.3.0` or later. For information, see Fargate platform versions for Amazon ECS.
  - To inject a specific JSON key or version of a secret as an environment variable or in a log configuration, you must use platform version `1.4.0` or later (Linux) or `1.0.0` (Windows). For information, see Fargate platform versions for Amazon ECS.
- For Amazon ECS tasks on EC2, the following should be considered:
  - To inject a secret using a specific JSON key or version of a secret, your container instance must have version `1.37.0` or later of the container agent. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS container agent.

    To inject the full contents of a secret as an environment variable or to inject a secret in a log configuration, your container instance must have version `1.22.0` or later of the container agent.
- Use interface VPC endpoints to enhance security controls and connect to Secrets Manager through a private subnet. You must create the interface VPC endpoints for Secrets Manager. For information about the VPC endpoint, see Create VPC endpoints in the *AWS Secrets Manager User*

Guide. For more information about using Secrets Manager and Amazon VPC, see How to connect to Secrets Manager service within a Amazon VPC.

- For Windows tasks that are configured to use the `awslogs` logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. Use the following syntax:

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers
 '["json-file","awslogs"]'
</powershell>
```

- Your task definition must use a task execution role with the additional permissions for Secrets Manager. For more information, see Amazon ECS task execution IAM role.

Create the AWS Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. For more information, see Create an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

Add the environment variable to the container definition

Within your container definition, you can specify the following:

- The `secrets` object containing the name of the environment variable to set in the container
- The Amazon Resource Name (ARN) of the Secrets Manager secret
- Additional parameters that contain the sensitive data to present to the container

The following example shows the full syntax that must be specified for the Secrets Manager secret.

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name:json-key:version-stage:version-id
```

The following section describes the additional parameters. These parameters are optional, but if you do not use them, you must include the colons `:` to use the default values. Examples are provided below for more context.

`json-key`

Specifies the name of the key in a key-value pair with the value that you want to set as the environment variable value. Only values in JSON format are supported. If you do not specify a JSON key, then the full contents of the secret is used.

`version-stage`

Specifies the staging label of the version of a secret that you want to use. If a version staging label is specified, you cannot specify a version ID. If no version stage is specified, the default behavior is to retrieve the secret with the `AWSCURRENT` staging label.

Staging labels are used to keep track of different versions of a secret when they are either updated or rotated. Each version of a secret has one or more staging labels and an ID.

`version-id`

Specifies the unique identifier of the version of a secret that you want to use. If a version ID is specified, you cannot specify a version staging label. If no version ID is specified, the default behavior is to retrieve the secret with the `AWSCURRENT` staging label.

Version IDs are used to keep track of different versions of a secret when they are either updated or rotated. Each version of a secret has an ID. For more information, see Key Terms and Concepts for AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

Example container definitions

The following examples show ways in which you can reference Secrets Manager secrets in your container definitions.

Example referencing a full secret

The following is a snippet of a task definition showing the format when referencing the full text of a Secrets Manager secret.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
    }]
  }]
```

```
}
```

To access the value of this secret from within the container you would need to call the `$environment_variable_name`.

Example referencing full secrets

The following is a snippet of a task definition showing the format when referencing the full text of multiple Secrets Manager secrets.

```
{
  "containerDefinitions": [{
     "secrets": [
      {
        "name": "environment_variable_name1",
         "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
      },
      {
        "name": "environment_variable_name2",
         "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-abcdef"
      },
      {
        "name": "environment_variable_name3",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-ABCDEF"
      }
    ]
  }]
}
```

To access the value of this secret from within the container you would need to call the `$environment_variable_name1`, `$environment_variable_name2`, and `$environment_variable_name3`.

Example referencing a specific key within a secret

The following shows an example output from a [get-secret-value](#) command that displays the contents of a secret along with the version staging label and version ID associated with it.

```
{
```

```
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981ddEXAMPLE",
    "SecretString": "{\"username1\":\"password1\",\"username2\":\"password2\",
\"username3\":\"password3\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1581968848.921
}
```

Reference a specific key from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::"
    }]
  }]
}
```

Example referencing a specific secret version

The following shows an example output from a describe-secret command that displays the unencrypted contents of a secret along with the metadata for all versions of the secret.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981ddEXAMPLE": [
            "AWSCURRENT"
        ],
        "9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE": [
            "AWSPREVIOUS"
```

```
        ]
    }
}
```

Reference a specific version staging label from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf::AWSPREVIOUS:"
    }]
  }]
}
```

Reference a specific version ID from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
    }]
  }]
}
```

Example referencing a specific key and version staging label of a secret

The following shows how to reference both a specific key within a secret and a specific version staging label.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf:username1:AWSPREVIOUS:"
    }]
```

```
  }]
}
```

To specify a specific key and version ID, use the following syntax.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
    }]
  }]
}
```

For information about how to create a task definition with the secret specified in an environment variable, see Creating an Amazon ECS task definition using the console.

Pass Systems Manager parameters through Amazon ECS environment variables

Amazon ECS allows you to inject sensitive data into your containers by storing your sensitive data in AWS Systems Manager Parameter Store parameters and then referencing them in your container definition.

Consider the following when using an environment variable to inject a Systems Manager secret into a container.

- Sensitive data is injected into your container when the container is initially started. If the secret is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- For Amazon ECS tasks on AWS Fargate, the following should be considered:
  - To inject the full content of a secret as an environment variable or in a log configuration, you must use platform version `1.3.0` or later. For information, see Fargate platform versions for Amazon ECS.
  - To inject a specific JSON key or version of a secret as an environment variable or in a log configuration, you must use platform version `1.4.0` or later (Linux) or `1.0.0` (Windows). For information, see Fargate platform versions for Amazon ECS.
- For Amazon ECS tasks on EC2, the following should be considered:

  - To inject a secret using a specific JSON key or version of a secret, your container instance must have version `1.37.0` or later of the container agent. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS container agent.

    To inject the full contents of a secret as an environment variable or to inject a secret in a log configuration, your container instance must have version `1.22.0` or later of the container agent.
- Use interface VPC endpoints to enhance security controls. You must create the interface VPC endpoints for Systems Manager. For information about the VPC endpoint, see Improve the security of EC2 instances by using VPC endpoints for Systems Manager in the *AWS Systems Manager User Guide*.
- Your task definition must use a task execution role with the additional permissions for Systems Manager Parameter Store. For more information, see Amazon ECS task execution IAM role.
- For Windows tasks that are configured to use the `awslogs` logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. Use the following syntax:

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers
 '["json-file","awslogs"]'
</powershell>
```

Create the Systems Manager parameter

You can use the Systems Manager console to create a Systems Manager Parameter Store parameter for your sensitive data. For more information, see Create a Systems Manager parameter (console) or Create a Systems Manager parameter (AWS CLI) in the *AWS Systems Manager User Guide*.

Add the environment variable to the container definition

Within your container definition in the task definition, specify `secrets` with the name of the environment variable to set in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container. For more information, see `secrets`.

The following is a snippet of a task definition showing the format when referencing a Systems Manager Parameter Store parameter. If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then specify the full ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }]
}
```

For information about how to create a task definition with the secret specified in an environment variable, see Creating an Amazon ECS task definition using the console.

Update your application to programmatically retrieve Systems Manager Parameter Store secrets

To retrieve the sensitive data stored in the Systems Manager Parameter Store parameter, see Code examples for Systems Manager using AWS SDKs in the *AWS SDK Code Examples Code Library*.

Pass secrets for Amazon ECS logging configuration

You can use the `secretOptions` parameter in `logConfiguration` to pass sensitive data used for logging.

You can store the secret in Secrets Manager or Systems Manager.

Use Secrets Manager

Within your container definition, when specifying a `logConfiguration` you can specify `secretOptions` with the name of the log driver option to set in the container and the full ARN of the Secrets Manager secret containing the sensitive data to present to the container. For more information about creating secrets, see Create an AWS Secrets Manager.

The following is a snippet of a task definition showing the format when referencing an Secrets Manager secret.

```
{
```

```
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "splunk",
      "options": {
        "splunk-url": "https://your_splunk_instance:8088"
      },
      "secretOptions": [{
        "name": "splunk-token",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
      }]
    }]
  }]
}
```

Add the environment variable to the container definition

Within your container definition, specify `secrets` with the name of the environment variable to set in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container. For more information, see [`secrets`](#).

The following is a snippet of a task definition showing the format when referencing a Systems Manager Parameter Store parameter. If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then specify the full ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }]
}
```

For information about how to create a task definition with the secret specified in an environment variable, see [Creating an Amazon ECS task definition using the console](#).

Use Systems Manager

You can inject sensitive data in a log configuration. Within your container definition, when specifying a `logConfiguration` you can specify `secretOptions` with the name of the log driver option to set in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

> **Important**
>
> If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then specify the full ARN.

The following is a snippet of a task definition showing the format when referencing a Systems Manager Parameter Store parameter.

```
{
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "fluentd",
      "options": {
        "tag": "fluentd demo"
      },
      "secretOptions": [{
        "name": "fluentd-address",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter:/parameter_name"
      }]
    }]
  }]
}
```

Specifying sensitive data using Secrets Manager secrets in Amazon ECS

Amazon ECS allows you to inject sensitive data into your containers by storing your sensitive data in AWS Secrets Manager secrets and then referencing them in your container definition. For more information, see Pass sensitive data to an Amazon ECS container.

Learn how to create an Secrets Manager secret, reference the secret in an Amazon ECS task definition, and then verify it worked by querying the environment variable inside a container showing the contents of the secret.

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in Set up to use Amazon ECS have been completed.
- Your user has the required IAM permissions to create the Secrets Manager and Amazon ECS resources.

Step 1: Create an Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. In this tutorial we will be creating a basic secret for storing a username and password to reference later in a container. For more information, see Create an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

The **key/value pairs to be stored in this secret** is the environment variable value in your container at the end of the tutorial.

Save the **Secret ARN** to reference in your task execution IAM policy and task definition in later steps.

Step 2: Add the secrets permissions to the task execution role

In order for Amazon ECS to retrieve the sensitive data from your Secrets Manager secret, you must have the secrets permissions for the task execution role. For more information, see Secrets Manager or Systems Manager permissions.

Step 3: Create a task definition

You can use the Amazon ECS console to create a task definition that references a Secrets Manager secret.

To create a task definition that specifies a secret

Use the IAM console to update your task execution role with the required permissions.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, enter the following task definition JSON text, ensuring that you specify the full ARN of the Secrets Manager secret you created in step 1 and the task execution role you updated in step 2. Choose **Save**.

5.
```
{
    "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
            ],
            "cpu": 10,
            "secrets": [
                {
                    "valueFrom":
 "arn:aws:secretsmanager:region:aws_account_id:secret:username_value",
                    "name": "username_value"
                }
            ],
            "memory": 300,
            "image": "public.ecr.aws/docker/library/httpd:2.4",
            "essential": true,
            "name": "ecs-secrets-container"
        }
    ],
    "family": "ecs-secrets-tutorial"
}
```

6. Choose **Create**.

Step 4: Create a cluster

You can use the Amazon ECS console to create a cluster containing a container instance to run the task on. If you have an existing cluster with at least one container instance registered to it with the available resources to run one instance of the task definition created for this tutorial you can skip to the next step.

For this tutorial we will be creating a cluster with one `t2.micro` container instance using the Amazon ECS-optimized Amazon Linux 2 AMI.

For information about how to create a cluster for the EC2 launch type, see the section called "Creating a cluster for the Amazon EC2 launch type".

Step 5: Run a task

You can use the Amazon ECS console to run a task using the task definition you created. For this tutorial we will be running a task using the EC2 launch type, using the cluster we created in the previous step.

For information about how to run a task, see the section called "Running an application as a task".

Step 6: Verify

You can verify all of the steps were completed successfully and the environment variable was created properly in your container using the following steps.

To verify that the environment variable was created

1. Find the public IP or DNS address for your container instance.
   a. Open the console at https://console.aws.amazon.com/ecs/v2.
   b. In the navigation pane, choose **Clusters**, and then choose the cluster you created.
   c. Choose **Infrastructure**, and then choose the container instance.
   d. Record the **Public IP** or **Public DNS** for your instance.
2. If you are using a macOS or Linux computer, connect to your instance with the following command, substituting the path to your private key and the public address for your instance:

```
$ ssh -i /path/to/my-key-pair.pem ec2-user@ec2-198-51-100-1.compute-1.amazonaws.com
```

For more information about using a Windows computer, see Connect to your Linux instance using PuTTY in the *Amazon EC2 User Guide*.

> **Important**
>
> For more information about any issues while connecting to your instance, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide*.

3. List the containers running on the instance. Note the container ID for `ecs-secrets-tutorial` container.

```
docker ps
```

4. Connect to the `ecs-secrets-tutorial` container using the container ID from the output of the previous step.

```
docker exec -it container_ID /bin/bash
```

5. Use the `echo` command to print the value of the environment variable.

```
echo $username_value
```

If the tutorial was successful, you should see the following output:

```
password_value
```

> **Note**
>
> Alternatively, you can list all environment variables in your container using the `env` (or `printenv`) command.

Step 7: Clean up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

To clean up the resources

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. Choose **Delete Cluster**.
5. In the confirmation box, enter **delete *`cluster name`***, and then choose **Delete**.
6. Open the IAM console at https://console.aws.amazon.com/iam/.
7. In the navigation pane, choose **Roles**.
8. Search the list of roles for `ecsTaskExecutionRole` and select it.
9. Choose **Permissions**, then choose the **X** next to **ECSSecretsTutorial**. Choose **Remove**.
10. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
11. Select the **username_value** secret you created and choose **Actions**, **Delete secret**.

Amazon ECS task definition parameters for the Fargate launch type

Task definitions are split into separate parts: the task family, the AWS Identity and Access Management (IAM) task role, the network mode, container definitions, volumes, and launch types. The family and container definitions are required in a task definition. In contrast, task role, network mode, volumes, and launch type are optional.

You can use these parameters in a JSON file to configure your task definition.

The following are more detailed descriptions for each task definition parameter for the Fargate launch type.

Family

`family`

Type: String

Required: Yes

When you register a task definition, you give it a family, which is similar to a name for multiple versions of the task definition, specified with a revision number. The first task definition that's registered into a particular family is given a revision of 1, and any task definitions registered after that are given a sequential revision number.

Launch types

When you register a task definition, you can specify a launch type that Amazon ECS should validate the task definition against. If the task definition doesn't validate against the compatibilities specified, a client exception is returned. For more information, see Amazon ECS launch types.

The following parameter is allowed in a task definition.

`requiresCompatibilities`

Type: String array

Required: No

Valid Values: `FARGATE`

The launch type to validate the task definition against. This initiates a check to ensure that all of the parameters that are used in the task definition meet the requirements of the launch type.

Task role

`taskRoleArn`

Type: String

Required: No

When you register a task definition, you can provide a task role for an IAM role that allows the containers in the task permission to call the AWS APIs that are specified in its associated policies on your behalf. For more information, see Amazon ECS task IAM role.

Task execution role

`executionRoleArn`

Type: String

Required: Conditional

The Amazon Resource Name (ARN) of the task execution role that grants the Amazon ECS container agent permission to make AWS API calls on your behalf.

> **Note**
>
> The task execution IAM role is required depending on the requirements of your task. For more information, see [Amazon ECS task execution IAM role](#).

Network mode

`networkMode`

Type: String

Required: Yes

The Docker networking mode to use for the containers in the task. For Amazon ECS tasks hosted on Fargate, the `awsvpc` network mode is required.

Runtime platform

`operatingSystemFamily`

Type: String

Required: Conditional

Default: LINUX

This parameter is required for Amazon ECS tasks that are hosted on Fargate.

When you register a task definition, you specify the operating system family.

The valid values are `LINUX`, `WINDOWS_SERVER_2025_FULL`, `WINDOWS_SERVER_2025_CORE`, `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`.

All task definitions that are used in a service must have the same value for this parameter.

When a task definition is part of a service, this value must match the service `platformFamily` value.

`cpuArchitecture`

Type: String

Required: Conditional

Default: X86_64

If the parameter is left as `null`, the default value is automatically assigned upon the initiation of a task hosted on Fargate.

When you register a task definition, you specify the CPU architecture. The valid values are `X86_64` and `ARM64`.

All task definitions that are used in a service must have the same value for this parameter.

When you have Linux tasks, you can set the value to `ARM64`. For more information, see the section called "Task definitions for 64-bit ARM workloads".

Task size

When you register a task definition, you can specify the total CPU and memory used for the task. This is separate from the `cpu` and `memory` values at the container definition level. For tasks that are hosted on Fargate (both Linux and Windows), these fields are required and there are specific values for both `cpu` and `memory` that are supported.

The following parameter is allowed in a task definition:

`cpu`

Type: String

Required: Yes

> **Note**
>
> Task-level CPU and memory parameters are required and used to determine the instance type and size that tasks run on. For Windows tasks, these values aren't

enforced at runtime, because Windows doesn't have a native mechanism that can easily enforce collective resource limits on a group of containers. If you want to enforce resource limits, we recommend using the container-level resources for Windows containers.

The hard limit of CPU units to present for the task. You can specify CPU values in the JSON file as a string in CPU units or virtual CPUs (vCPUs). For example, you can specify a CPU value either as `1024` in CPU units or `1 vCPU` in vCPUs. When the task definition is registered, a vCPU value is converted to an integer indicating the CPU units.

This field is required and you must use one of the following values, which determines your range of supported values for the `memory` parameter. The table below shows the valid combinations of task-level CPU and memory.

CPU value	Memory value	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) **Note** This option requires Linux platform `1.4.0` or later.	Between 16 GB and 60 GB in 4 GB increments	Linux

CPU value	Memory value	Operating systems supported for AWS Fargate
16384 (16vCPU) **Note** This option requires Linux platform `1.4.0` or later.	Between 32 GB and 120 GB in 8 GB increments	Linux

`memory`

Type: String

Required: Yes

> **Note**
>
> Task-level CPU and memory parameters are required and used to determine the instance type and size that tasks run on. For Windows tasks, these values aren't enforced at runtime, because Windows doesn't have a native mechanism that can easily enforce collective resource limits on a group of containers. If you want to enforce resource limits, we recommend using the container-level resources for Windows containers.

The hard limit of memory to present to the task. You can specify memory values in the task definition as a string in mebibytes (MiB) or gigabytes (GB). For example, you can specify a memory value either as `3072` in MiB or `3 GB`in GB. When the task definition is registered, a GB value is converted to an integer indicating the MiB.

This field is required and you must use one of the following values, which determines your range of supported values for the `cpu` parameter:

Memory value (in MiB, with approximate equivalent value in GB)	CPU value	Operating systems supported for Fargate
512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)	256 (.25 vCPU)	Linux
1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)	512 (.5 vCPU)	Linux
2048 (2 GB), 3072 (3 GB), 4096 (4GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)	1024 (1 vCPU)	Linux, Windows
Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)	2048 (2 vCPU)	Linux, Windows
Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)	4096 (4 vCPU)	Linux, Windows
Between 16 GB and 60 GB in 4 GB increments **Note** This option requires Linux platform `1.4.0` or later.	8192 (8 vCPU)	Linux
Between 32 GB and 120 GB in 8 GB increments	16384 (16vCPU)	Linux

Memory value (in MiB, with approximate equivalent value in GB)	CPU value	Operating systems supported for Fargate
Note This option requires Linux platform `1.4.0` or later.		

Container definitions

When you register a task definition, you must specify a list of container definitions that are passed to the Docker daemon on a container instance. The following parameters are allowed in a container definition.

Topics

Standard container definition parameters

The following task definition parameters are either required or used in most container definitions.

Topics

Name

`name`

Type: String

Required: Yes

The name of a container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. If you're linking multiple containers in a task definition, the `name` of one container can be entered in the `links` of another container. This is to connect the containers.

Image

`image`

Type: String

Required: Yes

The image used to start a container. This string is passed directly to the Docker daemon. By default, images in the Docker Hub registry are available. You can also specify other repositories with either `repository-url/image:tag` or `repository-url/image@digest`. Up to 255 letters (uppercase and lowercase), numbers, hyphens, underscores, colons, periods, forward slashes, and number signs are allowed. This parameter maps to `Image` in the docker create-container command and the `IMAGE` parameter of the docker run command.

- When a new task starts, the Amazon ECS container agent pulls the latest version of the specified image and tag for the container to use. However, subsequent updates to a repository image aren't propagated to already running tasks.
- Whenyou don't specify a tag or digest in the image path in the task definition, the Amazon ECS container agent pulls the latest version of the specified image.
- However, subsequent updates to a repository image aren't propagated to already running tasks.
- Images in private registries are supported. For more information, see Using non-AWS container images in Amazon ECS.
- Images in Amazon ECR repositories can be specified by using either the full `registry/repository:tag` or `registry/repository@digest` naming convention (for

example, *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*:*latest* or *aws_account_id*.dkr.ecr.*region*.amazonaws.com/*my-web-app*@*sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE*).

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

`versionConsistency`

Type: String

Valid values: `enabled`|`disabled`

Required: No

Specifies whether Amazon ECS will resolve the container image tag provided in the container definition to an image digest. By default, this behavior is `enabled`. If you set the value for a container as `disabled`, Amazon ECS will not resolve the container image tag to a digest and will use the original image URI specified in the container definition for deployment. For more information about container image resolution, see Container image resolution.

Memory

`memory`

Type: Integer

Required: No

The amount (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. The total amount of memory reserved for all containers within a task must be lower than the task `memory` value, if one is specified. This parameter maps to `Memory` in the docker create-container command and the `--memory` option to docker run.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

> **Note**
>
> If you're trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see Reserving Amazon ECS Linux container instance memory.

`memoryReservation`

Type: Integer

Required: No

The soft limit (in MiB) of memory to reserve for the container. When system memory is under contention, Docker attempts to keep the container memory to this soft limit. However, your container can use more memory when needed. The container can use up to the hard limit that's specified with the `memory` parameter (if applicable) or all of the available memory on the container instance, whichever comes first. This parameter maps to `MemoryReservation` in the docker create-container command and the `--memory-reservation` option to docker run.

If a task-level memory value isn't specified, you must specify a non-zero integer for one or both of `memory` or `memoryReservation` in a container definition. If you specify both, `memory` must be greater than `memoryReservation`. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance that the container is placed on. Otherwise, the value of `memory` is used.

For example, suppose that your container normally uses 128 MiB of memory, but occasionally bursts to 256 MiB of memory for short periods of time. You can set a `memoryReservation` of 128 MiB, and a `memory` hard limit of 300 MiB. This configuration allows the container to only reserve 128 MiB of memory from the remaining resources on the container instance. At the same time, this configuration also allows the container to use more memory resources when needed.

> **Note**
>
> This parameter isn't supported for Windows containers.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

> **Note**
>
> If you're trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see Reserving Amazon ECS Linux container instance memory.

Port mappings

`portMappings`

Type: Object array

Required: No

Port mappings expose your container's network ports to the outside world. this allows clients to access your application. It's also used for inter-container communication within the same task.

For task definitions that use the `awsvpc` network mode, only specify the `containerPort`. The `hostPort` is always ignored, and the container port is automatically mapped to a random high-numbered port on the host.

Port mappings on Windows use the `NetNAT` gateway address rather than `localhost`. There's no loopback for port mappings on Windows, so you can't access a container's mapped port from the host itself.

Most fields of this parameter (including `containerPort`, `hostPort`, `protocol`) map to `PortBindings` in thedocker create-container command and the `--publish` option to docker run. If the network mode of a task definition is set to `host`, host ports must either be undefined or match the container port in the port mapping.

> **Note**
>
> After a task reaches the `RUNNING` status, manual and automatic host and container port assignments are visible in the following locations:

- Console: The **Network Bindings** section of a container description for a selected task.
- AWS CLI: The `networkBindings` section of the **describe-tasks** command output.
- API: The `DescribeTasks` response.
- Metadata: The task metadata endpoint.

`appProtocol`

Type: String

Required: No

The application protocol that's used for the port mapping. This parameter only applies to Service Connect. We recommend that you set this parameter to be consistent with the protocol that your application uses. If you set this parameter, Amazon ECS adds protocol-specific connection handling to the service connect proxy. If you set this parameter, Amazon ECS adds protocol-specific telemetry in the Amazon ECS console and CloudWatch.

If you don't set a value for this parameter, then TCP is used. However, Amazon ECS doesn't add protocol-specific telemetry for TCP.

For more information, see [the section called " Service Connect"](#).

Valid protocol values: `"http" | "http2" | "grpc"`

`containerPort`

Type: Integer

Required: Yes, when `portMappings` are used

The port number on the container that's bound to the user-specified or automatically assigned host port.

For tasks that use the `awsvpc` network mode, you use `containerPort` to specify the exposed ports.

For Windows containers on Fargate, you can't use port 3150 for the `containerPort`. This is because it's reserved.

`containerPortRange`

Type: String

Required: No

The port number range on the container that's bound to the dynamically mapped host port range.

You can only set this parameter by using the `register-task-definition` API. The option is available in the `portMappings` parameter. For more information, see register-task-definition in the *AWS Command Line Interface Reference*.

The following rules apply when you specify a `containerPortRange`:

- You must use the `awsvpc` network mode.
- This parameter is available for both the Linux and Windows operating systems.
- The container instance must have at least version 1.67.0 of the container agent and at least version 1.67.0-1 of the `ecs-init` package.
- You can specify a maximum of 100 port ranges for each container.
- You don't specify a `hostPortRange`. The value of the `hostPortRange` is set as follows:
  - For containers in a task with the `awsvpc` network mode, the `hostPort` is set to the same value as the `containerPort`. This is a static mapping strategy.
- The `containerPortRange` valid values are between 1 and 65535.
- A port can only be included in one port mapping for each container.
- You can't specify overlapping port ranges.
- The first port in the range must be less than last port in the range.
- Docker recommends that you turn off the docker-proxy in the Docker daemon config file when you have a large number of ports.

  For more information, see Issue #11185 on GitHub.

  For information about how to turn off the docker-proxy in the Docker daemon config file, see Docker daemon in the *Amazon ECS Developer Guide*.

You can call DescribeTasks to view the `hostPortRange`, which are the host ports that are bound to the container ports.

The port ranges aren't included in the Amazon ECS task events, which are sent to EventBridge. For more information, see [the section called “Automate responses to Amazon ECS errors using EventBridge”](#).

`hostPortRange`

Type: String

Required: No

The port number range on the host that's used with the network binding. This is assigned by Docker and delivered by the Amazon ECS agent.

`hostPort`

Type: Integer

Required: No

The port number on the container instance to reserve for your container.

The `hostPort` can either be kept blank or be the same value as `containerPort`.

The default ephemeral port range Docker version 1.6.0 and later is listed on the instance under `/proc/sys/net/ipv4/ip_local_port_range`. If this kernel parameter is unavailable, the default ephemeral port range from `49153–65535` is used. Don't attempt to specify a host port in the ephemeral port range. This is because these are reserved for automatic assignment. In general, ports under `32768` are outside of the ephemeral port range.

The default reserved ports are 22 for SSH, the Docker ports `2375` and `2376`, and the Amazon ECS container agent ports `51678-51680`. Any host port that was previously user-specified for a running task is also reserved while the task is running. After a task stops, the host port is released. The current reserved ports are displayed in the `remainingResources` of **describe-container-instances** output. A container instance might have up to 100 reserved ports at a time, including the default reserved ports. Automatically assigned ports don't count toward the 100 reserved ports quota.

`name`

Type: String

Required: No, required for Service Connect and VPC Lattice to be configured in a service

The name that's used for the port mapping. This parameter only applies to Service Connect and VPC Lattice. This parameter is the name that you use in the Service Connect and VPC Lattice configuration of a service.

For more information, see Use Service Connect to connect Amazon ECS services with short names.

In the following example, both of the required fields for Service Connect and VPC Lattice are used.

```
"portMappings": [
    {
        "name": string,
        "containerPort": integer
    }
]
```

`protocol`

Type: String

Required: No

The protocol that's used for the port mapping. Valid values are `tcp` and `udp`. The default is `tcp`.

> **Important**
>
> Only `tcp` is supported for Service Connect. Remember that `tcp` is implied if this field isn't set.

If you're specifying a host port, use the following syntax.

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
```

```
    ...
]
```

If you want an automatically assigned host port, use the following syntax.

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
]
```

Private Repository Credentials

`repositoryCredentials`

Type: RepositoryCredentials object

Required: No

The repository credentials for private registry authentication.

For more information, see Using non-AWS container images in Amazon ECS.

`credentialsParameter`

Type: String

Required: Yes, when `repositoryCredentials` are used

The Amazon Resource Name (ARN) of the secret containing the private repository credentials.

For more information, see Using non-AWS container images in Amazon ECS.

> **Note**
>
> When you use the Amazon ECS API, AWS CLI, or AWS SDKs, if the secret exists in the same Region as the task that you're launching then you can use either the full ARN or the name of the secret. When you use the AWS Management Console, you must specify the full ARN of the secret.

The following is a snippet of a task definition that shows the required parameters:

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

Advanced container definition parameters

The following advanced container definition parameters provide extended capabilities to the docker run command that's used to launch containers on your Amazon ECS container instances.

Topics

Restart policy

`restartPolicy`

The container restart policy and associated configuration parameters. When you set up a restart policy for a container, Amazon ECS can restart the container without needing to replace the task. For more information, see Restart individual containers in Amazon ECS tasks with container restart policies.

`enabled`

Type: Boolean

Required: Yes

Specifies whether a restart policy is enabled for the container.

`ignoredExitCodes`

Type: Integer array

Required: No

A list of exit codes that Amazon ECS will ignore and not attempt a restart on. You can specify a maximum of 50 container exit codes. By default, Amazon ECS does not ignore any exit codes.

`restartAttemptPeriod`

Type: Integer

Required: No

A period of time (in seconds) that the container must run for before a restart can be attempted. A container can be restarted only once every `restartAttemptPeriod` seconds. If a container isn't able to run for this time period and exits early, it will not be restarted. You can set a minimum `restartAttemptPeriod` of 60 seconds and a maximum `restartAttemptPeriod` of 1800 seconds. By default, a container must run for 300 seconds before it can be restarted.

Health check

`healthCheck`

The container health check command and the associated configuration parameters for the container. For more information, see Determine Amazon ECS task health using container health checks.

`command`

A string array that represents the command that the container runs to determine if it's healthy. The string array can start with `CMD` to run the command arguments directly, or `CMD-`

`SHELL` to run the command with the container's default shell. If neither is specified, `CMD` is used.

When registering a task definition in the AWS Management Console, use a comma separated list of commands. These commands are converted to a string after the task definition is created. An example input for a health check is the following.

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

When registering a task definition using the AWS Management Console JSON panel, the AWS CLI, or the APIs, enclose the list of commands in brackets. An example input for a health check is the following.

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

An exit code of 0, with no `stderr` output, indicates success, and a non-zero exit code indicates failure.

`interval`

The period of time (in seconds) between each health check. You can specify between 5 and 300 seconds. The default value is 30 seconds.

`timeout`

The period of time (in seconds) to wait for a health check to succeed before it's considered a failure. You can specify between 2 and 60 seconds. The default value is 5 seconds.

`retries`

The number of times to retry a failed health check before the container is considered unhealthy. You can specify between 1 and 10 retries. The default value is three retries.

`startPeriod`

The optional grace period to provide containers time to bootstrap in before failed health checks count towards the maximum number of retries. You can specify a value between 0 and 300 seconds. By default, `startPeriod` is disabled.

If a health check succeeds within the `startPeriod`, then the container is considered healthy and any subsequent failures count toward the maximum number of retries.

Environment

`cpu`

Type: Integer

Required: No

The number of `cpu` units the Amazon ECS container agent reserves for the container. On Linux, this parameter maps to `CpuShares` in the [Create a container](#) section.

This field is optional for tasks that use the Fargate launch type. The total amount of CPU reserved for all the containers that are within a task must be lower than the task-level `cpu` value.

Linux containers share unallocated CPU units with other containers on the container instance with the same ratio as their allocated amount. For example, assume that you run a single-container task on a single-core instance type with 512 CPU units specified for that container. Moreover, that task is the only task running on the container instance. In this example, the container can use the full 1,024 CPU unit share at any given time. However, assume then that you launched another copy of the same task on that container instance. Each task is guaranteed a minimum of 512 CPU units when needed. Similarly, if the other container isn't using the remaining CPU, each container can float to higher CPU usage. However, if both tasks were 100% active all of the time, they are limited to 512 CPU units.

On Linux container instances, the Docker daemon on the container instance uses the CPU value to calculate the relative CPU share ratios for running containers. The minimum valid CPU share value that the Linux kernel allows is 2, and the maximum valid CPU share value that the Linux kernel allows is 262144. However, the CPU parameter isn't required, and you can use CPU values below two and above 262144 in your container definitions. For CPU values below two (including null) and above 262144, the behavior varies based on your Amazon ECS container agent version:

On Windows container instances, the CPU quota is enforced as an absolute quota. Windows containers only have access to the specified amount of CPU that's defined in the task definition. A null or zero CPU value is passed to Docker as `0`. Windows then interprets this value as 1% of one CPU.

For more examples, see [How Amazon ECS manages CPU and memory resources](#).

`gpu`

This parameter isn't supported for containers that are hosted on Fargate.

Type: ResourceRequirement object

Required: No

The number of physical GPUs that the Amazon ECS container agent reserves for the container. The number of GPUs reserved for all containers in a task must not exceed the number of available GPUs on the container instance the task is launched on. For more information, see Amazon ECS task definitions for GPU workloads.

`Elastic Inference accelerator`

This parameter isn't supported for containers that are hosted on Fargate.

Type: ResourceRequirement object

Required: No

For the `InferenceAccelerator` type, the `value` matches the `deviceName` for an `InferenceAccelerator` specified in a task definition. For more information, see the section called "Elastic Inference accelerator name".

`essential`

Type: Boolean

Required: No

Suppose that the `essential` parameter of a container is marked as `true`, and that container fails or stops for any reason. Then, all other containers that are part of the task are stopped. If the `essential` parameter of a container is marked as `false`, then its failure doesn't affect the rest of the containers in a task. If this parameter is omitted, a container is assumed to be essential.

All tasks must have at least one essential container. Suppose that you have an application that's composed of multiple containers. Then, group containers that are used for a common purpose into components, and separate the different components into multiple task definitions. For more information, see Architect your application for Amazon ECS.

```
"essential": true|false
```

`entryPoint`

> **Important**
>
> Early versions of the Amazon ECS container agent don't properly handle `entryPoint` parameters. If you have problems using `entryPoint`, update your container agent or enter your commands and arguments as `command` array items instead.

Type: String array

Required: No

The entry point that's passed to the container.

```
"entryPoint": ["string", ...]
```

`command`

Type: String array

Required: No

The command that's passed to the container. This parameter maps to `Cmd` in the create-container command and the `COMMAND` parameter to docker run. If there are multiple arguments, make sure that each argument is a separated string in the array.

```
"command": ["string", ...]
```

`workingDirectory`

Type: String

Required: No

The working directory to run commands inside the container in. This parameter maps to `WorkingDir` in the [Create a container](#) section of the [Docker Remote API](#) and the `--workdir` option to [**docker run**](#).

```
"workingDirectory": "string"
```

`environmentFiles`

This isn't available for Windows containers on Fargate.

Type: Object array

Required: No

A list of files containing the environment variables to pass to a container. This parameter maps to the `--env-file` option to the docker run command.

You can specify up to 10 environment files. The file must have a `.env` file extension. Each line in an environment file contains an environment variable in `VARIABLE=VALUE` format. Lines that start with # are treated as comments and are ignored.

If there are individual environment variables specified in the container definition, they take precedence over the variables contained within an environment file. If multiple environment files are specified that contain the same variable, they're processed from the top down. We recommend that you use unique variable names. For more information, see Pass an individual environment variable to an Amazon ECS container.

`value`

Type: String

Required: Yes

The Amazon Resource Name (ARN) of the Amazon S3 object containing the environment variable file.

`type`

Type: String

Required: Yes

The file type to use. The only supported value is `s3`.

`environment`

Type: Object array

Required: No

The environment variables to pass to a container. This parameter maps to `Env` in the docker create-container command and the `--env` option to the docker run command.

> **Important**
>
> We do not recommend using plaintext environment variables for sensitive information, such as credential data.

`name`

Type: String

Required: Yes, when `environment` is used

The name of the environment variable.

`value`

Type: String

Required: Yes, when `environment` is used

The value of the environment variable.

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

`secrets`

Type: Object array

Required: No

An object that represents the secret to expose to your container. For more information, see [Pass sensitive data to an Amazon ECS container](#).

`name`

Type: String

Required: Yes

The value to set as the environment variable on the container.

`valueFrom`

Type: String

Required: Yes

The secret to expose to the container. The supported values are either the full Amazon Resource Name (ARN) of the AWS Secrets Manager secret or the full ARN of the parameter in the AWS Systems Manager Parameter Store.

> **Note**
>
> If the Systems Manager Parameter Store parameter or Secrets Manager parameter exists in the same AWS Region as the task that you're launching, you can use either the full ARN or name of the secret. If the parameter exists in a different Region, then the full ARN must be specified.

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

Network settings

`disableNetworking`

This parameter is not supported for tasks running on Fargate.

Type: Boolean

Required: No

When this parameter is true, networking is off within the container.

The default is `false`.

```
"disableNetworking": true|false
```

`links`

This parameter isn't supported for tasks using the `awsvpc` network mode.

Type: String array

Required: No

The `link` parameter allows containers to communicate with each other without the need for port mappings. This parameter is only supported if the network mode of a task definition is set to `bridge`. The `name:internalName` construct is analogous to `name:alias` in Docker links. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed..

> **Important**
>
> Containers that are collocated on the same container instance might communicate with each other without requiring links or host port mappings. The network isolation on a container instance is controlled by security groups and VPC settings.

```
"links": ["name:internalName", ...]
```

`hostname`

Type: String

Required: No

The hostname to use for your container. This parameter maps to `Hostname` in the docker create-container and the `--hostname` option to docker run.

> **Note**
>
> If you're using the `awsvpc` network mode, the `hostname` parameter isn't supported.

```
"hostname": "string"
```

`dnsServers`

This is not supported for tasks running on Fargate.

Type: String array

Required: No

A list of DNS servers that are presented to the container.

```
"dnsServers": ["string", ...]
```

`extraHosts`

This parameter isn't supported for tasks that use the `awsvpc` network mode.

Type: Object array

Required: No

A list of hostnames and IP address mappings to append to the `/etc/hosts` file on the container.

This parameter maps to `ExtraHosts` in the docker create-container command and the `--add-host` option to docker run.

```
"extraHosts": [
      {
        "hostname": "string",
        "ipAddress": "string"
      }
      ...
    ]
```

`hostname`

Type: String

Required: Yes, when `extraHosts` are used

The hostname to use in the `/etc/hosts` entry.

`ipAddress`

Type: String

Required: Yes, when `extraHosts` are used

The IP address to use in the `/etc/hosts` entry.

Storage and logging

`readonlyRootFilesystem`

Type: Boolean

Required: No

When this parameter is true, the container is given read-only access to its root file system. This parameter maps to `ReadonlyRootfs` in the docker create-container command the `--read-only` option to docker run.

> **Note**
>
> This parameter is not supported for Windows containers.

The default is `false`.

```
"readonlyRootFilesystem": true|false
```

`mountPoints`

Type: Object array

Required: No

The mount points for the data volumes in your container. This parameter maps to `Volumes` in the create-container Docker API and the `--volume` option to docker run.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount points cannot be used across drives. You must specify mount points to attach an Amazon EBS volume directly to an Amazon ECS task.

`sourceVolume`

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

`containerPath`

Type: String

Required: Yes, when `mountPoints` are used

The path in the container where the volume will be mounted.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

For tasks that run on EC2 instances running the Windows operating system, leave the value as the default of `false`.

`volumesFrom`

Type: Object array

Required: No

Data volumes to mount from another container. This parameter maps to `VolumesFrom` in the docker create-container command and the `--volumes-from` option to docker run.

`sourceContainer`

Type: String

Required: Yes, when `volumesFrom` is used

The name of the container to mount volumes from.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

```
"volumesFrom": [
                {
                  "sourceContainer": "string",
                  "readOnly": true|false
                }
              ]
```

`logConfiguration`

Type: LogConfiguration Object

Required: No

The log configuration specification for the container.

For example task definitions that use a log configuration, see Example Amazon ECS task definitions.

This parameter maps to `LogConfig` in the docker create-container command and the `--log-driver` option to docker run. By default, containers use the same logging driver that the Docker daemon uses. However, the container might use a different logging driver than the Docker daemon by specifying a log driver with this parameter in the container definition. To use a different logging driver for a container, the log system must be configured properly on the container instance (or on a different log server for remote logging options).

Consider the following when specifying a log configuration for your containers:

- Amazon ECS supports a subset of the logging drivers that are available to the Docker daemon. Additional log drivers might be available in future releases of the Amazon ECS container agent.

- This parameter requires version 1.18 or later of the Docker Remote API on your container instance.
- You must install any additional software outside of the task. For example, the Fluentd output aggregators or a remote host running Logstash to send Gelf logs to.

```
"logConfiguration": {
      "logDriver": "awslogs","fluentd","gelf","json-
file","journald","splunk","syslog","awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
  "name": "string",
  "valueFrom": "string"
 }]
}
```

`logDriver`

Type: String

Valid values: `"awslogs","fluentd","gelf","json-file","journald","splunk","syslog","awsfirelens"`

Required: Yes, when `logConfiguration` is used

The log driver to use for the container. By default, the valid values that are listed earlier are log drivers that the Amazon ECS container agent can communicate with.

The supported log drivers are `awslogs`, `splunk`, and `awsfirelens`.

For more information about how to use the `awslogs` log driver in task definitions to send your container logs to CloudWatch Logs, see Send Amazon ECS logs to CloudWatch .

For more information about using the `awsfirelens` log driver, see Custom Log Routing.

> **Note**
>
> If you have a custom driver that isn't listed, you can fork the Amazon ECS container agent project that's available on GitHub and customize it to work with that driver. We encourage you to submit pull requests for changes that you want to have

included. However, we don't currently support running modified copies of this software.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

`options`

Type: String to string map

Required: No

The key/value map of configuration options to send to the log driver.

The options you can specify depend on the log driver. Some of the options you can specify when you use the `awslogs` router to route logs to Amazon CloudWatch include the following:

`awslogs-create-group`

Required: No

Specify whether you want the log group to be created automatically. If this option isn't specified, it defaults to `false`.

Note

Your IAM policy must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`.

`awslogs-region`

Required: Yes

Specify the AWS Region that the `awslogs` log driver is to send your Docker logs to. You can choose to send all of your logs from clusters in different Regions to a single region in CloudWatch Logs. This is so that they're all visible in one location. Otherwise, you can separate them by Region for more granularity. Make sure that the specified log group exists in the Region that you specify with this option.

`awslogs-group`

Required: Yes

Make sure to specify a log group that the `awslogs` log driver sends its log streams to.

`awslogs-stream-prefix`

Required: Yes

Use the `awslogs-stream-prefix` option to associate a log stream with the specified prefix, the container name, and the ID of the Amazon ECS task that the container belongs to. If you specify a prefix with this option, then the log stream takes the following format.

```
prefix-name/container-name/ecs-task-id
```

If you don't specify a prefix with this option, then the log stream is named after the container ID that's assigned by the Docker daemon on the container instance. Because it's difficult to trace logs back to the container that sent them with just the Docker container ID (which is only available on the container instance), we recommend that you specify a prefix with this option.

For Amazon ECS services, you can use the service name as the prefix. Doing so, you can trace log streams to the service that the container belongs to, the name of the container that sent them, and the ID of the task that the container belongs to.

You must specify a stream-prefix for your logs to have your logs appear in the Log pane when using the Amazon ECS console.

`awslogs-datetime-format`

Required: No

This option defines a multiline start pattern in Python `strftime` format. A log message consists of a line that matches the pattern and any following lines that don’t match the pattern. The matched line is the delimiter between log messages.

One example of a use case for using this format is for parsing output such as a stack dump, which might otherwise be logged in multiple entries. The correct pattern allows it to be captured in a single entry.

For more information, see awslogs-datetime-format.

You cannot configure both the `awslogs-datetime-format` and `awslogs-multiline-pattern` options.

> **Note**
>
> Multiline logging performs regular expression parsing and matching of all log messages. This might have a negative impact on logging performance.

`awslogs-multiline-pattern`

Required: No

This option defines a multiline start pattern that uses a regular expression. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. The matched line is the delimiter between log messages.

For more information, see awslogs-multiline-pattern.

This option is ignored if `awslogs-datetime-format` is also configured.

You cannot configure both the `awslogs-datetime-format` and `awslogs-multiline-pattern` options.

> **Note**
>
> Multiline logging performs regular expression parsing and matching of all log messages. This might have a negative impact on logging performance.

The following options apply to all supported log drivers.

`mode`

Required: No

Valid values: `non-blocking` | `blocking`

This option defines the delivery mode of log messages from the container to the log driver specified using `logDriver`. The delivery mode you choose affects application availability when the flow of logs from the container is interrupted.

If you use the `blocking` mode and the flow of logs is interrupted, calls from container code to write to the `stdout` and `stderr` streams will block. The logging thread of the application will block as a result. This may cause the application to become unresponsive and lead to container healthcheck failure.

If you use the `non-blocking` mode, the container's logs are instead stored in an in-memory intermediate buffer configured with the `max-buffer-size` option. This prevents the application from becoming unresponsive when logs cannot be sent. We recommend using this mode if you want to ensure service availability and are okay with some log loss. For more information, see [Preventing log loss with non-blocking mode in the `awslogs` container log driver](#).

You can set a default `mode` for all containers in a specific AWS Region by using the `defaultLogDriverMode` account setting. If you don't specify the `mode` option in the `logConfiguration` or configure the account setting, Amazon ECS will default to `non-blocking` mode. For more information about the account setting, see [Default log driver mode](#).

When `non-blocking` mode is used, the `max-buffer-size` log option controls the size of the buffer that's used for intermediate message storage. Make sure to specify an adequate buffer size based on your application. The total amount of memory allocated at the task level should be greater than the amount of memory that's allocated for all the containers in addition to the log driver memory buffer.

> **Note**
>
> On June 25, 2025, Amazon ECS changed the default log driver mode from `blocking` to `non-blocking` to prioritize task availability over logging. To continue using the `blocking` mode after this change, do one of the following:
>
> - Set the `mode` option in your container definition's `logConfiguration` as `blocking`.
> - Set the `defaultLogDriverMode` account setting to `blocking`.

`max-buffer-size`

Required: No

Default value: `10m`

When `non-blocking` mode is used, the `max-buffer-size` log option controls the size of the buffer that's used for intermediate message storage. Make sure to specify an adequate buffer size based on your application. When the buffer fills up, further logs cannot be stored. Logs that cannot be stored are lost.

To route logs using the `splunk` log router, you need to specify a `splunk-token` and a `splunk-url`.

When you use the `awsfirelens` log router to route logs to an AWS service or AWS Partner Network destination for log storage and analytics, you can set the `log-driver-buffer-limit` option to limit the number of events that are buffered in memory, before being sent to the log router container. It can help to resolve potential log loss issue because high throughput might result in memory running out for the buffer inside of Docker. For more information, see [the section called "Configuring logs for high throughput"].

Other options you can specify when using `awsfirelens` to route logs depend on the destination. When you export logs to Amazon Data Firehose, you can specify the AWS Region with `region` and a name for the log stream with `delivery_stream`.

When you export logs to Amazon Kinesis Data Streams, you can specify an AWS Region with `region` and a data stream name with `stream`.

When you export logs to Amazon OpenSearch Service, you can specify options like `Name`, `Host` (OpenSearch Service endpoint without protocol), `Port`, `Index`, `Type`, `Aws_auth`, `Aws_region`, `Suppress_Type_Name`, and `tls`.

When you export logs to Amazon S3, you can specify the bucket using the `bucket` option. You can also specify `region`, `total_file_size`, `upload_timeout`, and `use_put_object` as options.

This parameter requires version 1.19 of the Docker Remote API or greater on your container instance.

`secretOptions`

Type: Object array

Required: No

An object that represents the secret to pass to the log configuration. Secrets that are used in log configuration can include an authentication token, certificate, or encryption key. For more information, see [Pass sensitive data to an Amazon ECS container].

`name`

Type: String

Required: Yes

The value to set as the environment variable on the container.

`valueFrom`

Type: String

Required: Yes

The secret to expose to the log configuration of the container.

```
"logConfiguration": {
 "logDriver": "splunk",
 "options": {
  "splunk-url": "https://cloud.splunk.com:8080",
  "splunk-token": "...",
  "tag": "...",
  ...
 },
 "secretOptions": [{
  "name": "splunk-token",
  "valueFrom": "/ecs/logconfig/splunkcred"
 }]
}
```

`firelensConfiguration`

Type: FirelensConfiguration Object

Required: No

The FireLens configuration for the container. This is used to specify and configure a log router for container logs. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.

```
{
    "firelensConfiguration": {
        "type": "fluentd",
```

```
        "options": {
            "KeyName": ""
        }
    }
}
```

`options`

Type: String to string map

Required: No

The key/value map of options to use when configuring the log router. This field is optional and can be used to specify a custom configuration file or to add additional metadata, such as the task, task definition, cluster, and container instance details to the log event. If specified, the syntax to use is `"options":{"enable-ecs-log-metadata":"true|false","config-file-type:"s3|file","config-file-value":"arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf|filepath"}`. For more information, see Example Amazon ECS task definition: Route logs to FireLens.

`type`

Type: String

Required: Yes

The log router to use. The valid values are `fluentd` or `fluentbit`.

Security

For more information about container security, see Amazon ECS task and container security best practices.

`credentialSpecs`

Type: String array

Required: No

A list of ARNs in SSM or Amazon S3 to a credential spec (`CredSpec`) file that configures the container for Active Directory authentication. We recommend that you use this parameter instead of the `dockerSecurityOptions`. The maximum number of ARNs is 1.

There are two formats for each ARN.

credentialspecdomainless:MyARN

You use `credentialspecdomainless:MyARN` to provide a `CredSpec` with an additional section for a secret in Secrets Manager. You provide the login credentials to the domain in the secret.

Each task that runs on any container instance can join different domains.

You can use this format without joining the container instance to a domain.

credentialspec:MyARN

You use `credentialspec:MyARN` to provide a `CredSpec` for a single domain.

You must join the container instance to the domain before you start any tasks that use this task definition.

In both formats, replace `MyARN` with the ARN in SSM or Amazon S3.

The `credspec` must provide a ARN in Secrets Manager for a secret containing the username, password, and the domain to connect to. For better security, the instance isn't joined to the domain for domainless authentication. Other applications on the instance can't use the domainless credentials. You can use this parameter to run tasks on the same instance, even it the tasks need to join different domains. For more information, see Using gMSAs for Windows Containers and Using gMSAs for Linux Containers.

`user`

Type: String

Required: No

The user to use inside the container. This parameter maps to `User` in the docker create-container command and the `--user` option to docker run.

> **Important**
>
> When running tasks that use the `host` network mode, don't run containers using the root user (UID 0). As a security best practice, always use a non-root user.

You can specify the `user` using the following formats. If specifying a UID or GID, you must specify it as a positive integer.

- `user`
- `user:group`
- `uid`
- `uid:gid`
- `user:gid`
- `uid:group`

> **Note**
>
> This parameter is not supported for Windows containers.

```
"user": "string"
```

Resource limits

`ulimits`

Type: Object array

Required: No

A list of `ulimit` values to define for a container. This value overwrites the default resource quota setting for the operating system. This parameter maps to `Ulimits` in the docker create-container command and the `--ulimit` option to docker run.

Amazon ECS tasks hosted on Fargate use the default resource limit values set by the operating system with the exception of the `nofile` resource limit parameter. The `nofile` resource limit sets a restriction on the number of open files that a container can use. On Fargate, the default `nofile` soft limit is `65535` and hard limit is `65535`. You can set the values of both limits up to `1048576`. For more information, see [Task resource limits](#).

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

> **Note**
>
> This parameter is not supported for Windows containers.

```
"ulimits": [
      {
        "name":
 "core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|"rtp
        "softLimit": integer,
        "hardLimit": integer
      }
      ...
    ]
```

name

Type: String

Valid values: `"core" | "cpu" | "data" | "fsize" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"`

Required: Yes, when `ulimits` are used

The `type` of the `ulimit`.

`hardLimit`

Type: Integer

Required: Yes, when `ulimits` are used

The hard limit for the `ulimit` type. The value can be specified in bytes, seconds, or as a count, depending on the `type` of the `ulimit`.

`softLimit`

Type: Integer

Required: Yes, when `ulimits` are used

The soft limit for the `ulimit` type. The value can be specified in bytes, seconds, or as a count, depending on the `type` of the `ulimit`.

Docker labels

`dockerLabels`

Type: String to string map

Required: No

A key/value map of labels to add to the container. This parameter maps to `Labels` in the docker create-container command and the `--label` option to docker run.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

```
"dockerLabels": {"string": "string"
      ...}
```

Other container definition parameters

The following container definition parameters can be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see Creating an Amazon ECS task definition using the console.

Topics

- Linux parameters
- Container dependency
- Container timeouts
- System controls
- Interactive
- Pseudo terminal

Linux parameters

`linuxParameters`

Type: LinuxParameters object

Required: No

Linux-specific options that are applied to the container, such as KernelCapabilities.

> **Note**
>
> This parameter isn't supported for Windows containers.

```
"linuxParameters": {
      "capabilities": {
        "add": ["string", ...],
        "drop": ["string", ...]
        }
      }
```

`capabilities`

Type: KernelCapabilities object

Required: No

The Linux capabilities for the container that are dropped from the default configuration provided by Docker. For more information about these Linux capabilities, see the capabilities(7) Linux manual page.

`add`

Type: String array

Valid values: `"SYS_PTRACE"`

Required: No

The Linux capabilities for the container to add to the default configuration that's provided by Docker. This parameter maps to `CapAdd` in the docker create-container command and the `--cap-add` option to docker run.

`drop`

Type: String array

Valid values: `"ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER"`

```
| "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" |
"LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" |
"NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW"
| "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" |
"SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT"
| "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" |
"SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"
```

Required: No

The Linux capabilities for the container to remove from the default configuration that's provided by Docker. This parameter maps to `CapDrop` in the docker create-container command and the `--cap-drop` option to docker run.

`devices`

Any host devices to expose to the container. This parameter maps to `Devices` in the docker create-container command and the `--device` option to docker run.

> **Note**
>
> The `devices` parameter isn't supported when you use the Fargate launch type.

Type: Array of [Device](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_Device.html) objects

Required: No

`hostPath`

The path for the device on the host container instance.

Type: String

Required: Yes

`containerPath`

The path inside the container to expose the host device at.

Type: String

Required: No

`permissions`

The explicit permissions to provide to the container for the device. By default, the container has permissions for `read`, `write`, and `mknod` on the device.

Type: Array of strings

Valid Values: `read` | `write` | `mknod`

`initProcessEnabled`

Run an `init` process inside the container that forwards signals and reaps processes. This parameter maps to the `--init` option to docker run.

This parameter requires version 1.25 of the Docker Remote API or greater on your container instance.

`maxSwap`

This is not supported for tasks running on Fargate.

The total amount of swap memory (in MiB) a container can use. This parameter is translated to the `--memory-swap` option to docker run where the value is the sum of the container memory plus the `maxSwap` value.

If a `maxSwap` value of `0` is specified, the container doesn't use swap. Accepted values are `0` or any positive integer. If the `maxSwap` parameter is omitted, the container uses the swap configuration for the container instance that it's running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

`sharedMemorySize`

The value for the size (in MiB) of the `/dev/shm` volume. This parameter maps to the `--shm-size` option to docker run.

> **Note**
>
> If you're using tasks that use the Fargate launch type, the `sharedMemorySize` parameter isn't supported.

Type: Integer

`tmpfs`

The container path, mount options, and maximum size (in MiB) of the tmpfs mount. This parameter maps to the `--tmpfs` option to docker run.

> **Note**
>
> If you're using tasks that use the Fargate launch type, the `tmpfs` parameter isn't supported.

Type: Array of Tmpfs objects

Required: No

`containerPath`

The absolute file path where the tmpfs volume is to be mounted.

Type: String

Required: Yes

`mountOptions`

The list of tmpfs volume mount options.

Type: Array of strings

Required: No

Valid Values: `"defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"`

`size`

The maximum size (in MiB) of the tmpfs volume.

Type: Integer

Required: Yes

Container dependency

`dependsOn`

Type: Array of ContainerDependency objects

Required: No

The dependencies defined for container startup and shutdown. A container can contain multiple dependencies. When a dependency is defined for container startup, for container shutdown it is reversed. For an example, see Container dependency.

> **Note**
>
> If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

This parameter requires that the task or service uses platform version `1.3.0` or later (Linux) or `1.0.0` (Windows).

```
"dependsOn": [
    {
        "containerName": "string",
        "condition": "string"
    }
]
```

`containerName`

Type: String

Required: Yes

The container name that must meet the specified condition.

`condition`

Type: String

Required: Yes

The dependency condition of the container. The following are the available conditions and their behavior:

- `START` – This condition emulates the behavior of links and volumes today. The condition validates that a dependent container is started before permitting other containers to start.
- `COMPLETE` – This condition validates that a dependent container runs to completion (exits) before permitting other containers to start. This can be useful for non-essential containers that run a script and then exit. This condition can't be set on an essential container.
- `SUCCESS` – This condition is the same as `COMPLETE`, but it also requires that the container exits with a `zero` status. This condition can't be set on an essential container.
- `HEALTHY` – This condition validates that the dependent container passes its container health check before permitting other containers to start. This requires that the dependent container has health checks configured in the task definition. This condition is confirmed only at task startup.

Container timeouts

`startTimeout`

Type: Integer

Required: No

Example values: `120`

Time duration (in seconds) to wait before giving up on resolving dependencies for a container.

For example, you specify two containers in a task definition with `containerA` having a dependency on `containerB` reaching a `COMPLETE`, `SUCCESS`, or `HEALTHY` status. If a `startTimeout` value is specified for `containerB` and it doesn't reach the desired status within that time, then `containerA` doesn't start.

> **Note**
>
> If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

This parameter requires that the task or service uses platform version `1.3.0` or later (Linux). The maximum value is 120 seconds.

`stopTimeout`

Type: Integer

Required: No

Example values: `120`

Time duration (in seconds) to wait before the container is forcefully killed if it doesn't exit normally on its own.

This parameter requires that the task or service uses platform version `1.3.0` or later (Linux). If the parameter isn't specified, then the default value of 30 seconds is used. The maximum value is 120 seconds.

System controls

`systemControls`

Type: SystemControl object

Required: No

A list of namespace kernel parameters to set in the container. This parameter maps to `Sysctls` in the docker create-container commandand the `--sysctl` option to docker run. For example, you can configure `net.ipv4.tcp_keepalive_time` setting to maintain longer lived connections.

We don't recommend that you specify network-related `systemControls` parameters for multiple containers in a single task that also uses either the `awsvpc` or `host` network mode. Doing this has the following disadvantages:

- If you set `systemControls` for any container, it applies to all containers in the task. If you set different `systemControls` for multiple containers in a single task, the container that's started last determines which `systemControls` take effect.

If you're setting an IPC resource namespace to use for the containers in the task, the following conditions apply to your system controls. For more information, see IPC mode.

- For tasks that use the `host` IPC mode, IPC namespace `systemControls` aren't supported.
- For tasks that use the `task` IPC mode, IPC namespace `systemControls` values apply to all containers within a task.

> **Note**
>
> This parameter is not supported for Windows containers.

> **Note**
>
> This parameter is only supported for tasks that are hosted on AWS Fargate if the tasks are using platform version `1.4.0` or later (Linux). This isn't supported for Windows containers on Fargate.

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
    }
]
```

`namespace`

Type: String

Required: No

The namespace kernel parameter to set a `value` for.

Valid IPC namespace values: `"kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced"`, and `Sysctls` that start with `"fs.mqueue.*"`

Valid network namespace values: `Sysctls` that start with `"net.*"`. On Fargate, only namespaced `Sysctls` that exist within the container are accepted.

All of these values are supported by Fargate.

`value`

Type: String

Required: No

The value for the namespace kernel parameter that's specified in `namespace`.

Interactive

`interactive`

Type: Boolean

Required: No

When this parameter is `true`, you can deploy containerized applications that require `stdin` or a `tty` to be allocated. This parameter maps to `OpenStdin` in the docker create-container command and the `--interactive` option to docker run.

The default is `false`.

Pseudo terminal

`pseudoTerminal`

Type: Boolean

Required: No

When this parameter is `true`, a TTY is allocated. This parameter maps to `Tty` in the docker create-container command and the `--tty` option to docker run.

The default is `false`.

Elastic Inference accelerator name

The Elastic Inference accelerator resource requirement for your task definition.

> **Note**
>
> Amazon Elastic Inference (EI) is no longer available to customers.

The following parameters are allowed in a task definition:

`deviceName`

Type: String

Required: Yes

The Elastic Inference accelerator device name. The `deviceName` must also be referenced in a container definition see `Elastic Inference accelerator`.

`deviceType`

Type: String

Required: Yes

The Elastic Inference accelerator to use.

Proxy configuration

`proxyConfiguration`

Type: ProxyConfiguration object

Required: No

The configuration details for the App Mesh proxy.

> **Note**
>
> This parameter is not supported for Windows containers.

```
"proxyConfiguration": {
```

```
    "type": "APPMESH",
    "containerName": "string",
    "properties": [
        {
            "name": "string",
            "value": "string"
        }
    ]
}
```

`type`

Type: String

Value values: `APPMESH`

Required: No

The proxy type. The only supported value is `APPMESH`.

`containerName`

Type: String

Required: Yes

The name of the container that serves as the App Mesh proxy.

`properties`

Type: Array of KeyValuePair objects

Required: No

The set of network configuration parameters to provide the Container Network Interface (CNI) plugin, specified as key-value pairs.

- `IgnoredUID` – (Required) The user ID (UID) of the proxy container as defined by the `user` parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredGID` is specified, this field can be empty.
- `IgnoredGID` – (Required) The group ID (GID) of the proxy container as defined by the `user` parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredUID` is specified, this field can be empty.

- `AppPorts` – (Required) The list of ports that the application uses. Network traffic to these ports is forwarded to the `ProxyIngressPort` and `ProxyEgressPort`.
- `ProxyIngressPort` – (Required) Specifies the port that incoming traffic to the `AppPorts` is directed to.
- `ProxyEgressPort` – (Required) Specifies the port that outgoing traffic from the `AppPorts` is directed to.
- `EgressIgnoredPorts` – (Required) The outbound traffic going to these specified ports is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.
- `EgressIgnoredIPs` – (Required) The outbound traffic going to these specified IP addresses is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.

`name`

Type: String

Required: No

The name of the key-value pair.

`value`

Type: String

Required: No

The value of the key-value pair.

Volumes

When you register a task definition, you can optionally specify a list of volumes to be passed to the Docker daemon on a container instance, which then becomes available for access by other containers on the same container instance.

The following are the types of data volumes that can be used:

- Amazon EBS volumes — Provides cost-effective, durable, high-performance block storage for data intensive containerized workloads. You can attach 1 Amazon EBS volume per Amazon ECS task when running a standalone task, or when creating or updating a service. Amazon EBS volumes are supported for Linux tasks hosted on Fargate. For more information, see Use Amazon EBS volumes with Amazon ECS.

- Amazon EFS volumes — Provides simple, scalable, and persistent file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic. It grows and shrinks automatically as you add and remove files. Your applications can have the storage that they need and when they need it. Amazon EFS volumes are supported for tasks that are hosted on Fargate. For more information, see Use Amazon EFS volumes with Amazon ECS.
- FSx for Windows File Server volumes — Provides fully managed Microsoft Windows file servers. These file servers are backed by a Windows file system. When using FSx for Windows File Server together with Amazon ECS, you can provision your Windows tasks with persistent, distributed, shared, and static file storage. For more information, see Use FSx for Windows File Server volumes with Amazon ECS.

  Windows containers on Fargate do not support this option.
- Bind mounts – A file or directory on the host machine that is mounted into a container. Bind mount host volumes are supported when running tasks. To use bind mount host volumes, specify a `host` and optional `sourcePath` value in your task definition.

For more information, see Storage options for Amazon ECS tasks.

The following parameters are allowed in a container definition.

`name`

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens (-), and underscores (_) are allowed. This name is referenced in the `sourceVolume` parameter of the container definition `mountPoints` object.

`host`

Required: No

The `host` parameter is used to tie the lifecycle of the bind mount to the host Amazon EC2 instance, rather than the task, and where it is stored. If the `host` parameter is empty, then the Docker daemon assigns a host path for your data volume, but the data is not guaranteed to persist after the containers associated with it stop running.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`.

> **Note**
>
> The `sourcePath` parameter is supported only when using tasks that are hosted on Amazon EC2 instances.

`sourcePath`

Type: String

Required: No

When the `host` parameter is used, specify a `sourcePath` to declare the path on the host Amazon EC2 instance that is presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you. If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host Amazon EC2 instance until you delete it manually. If the `sourcePath` value does not exist on the host Amazon EC2 instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

`configuredAtLaunch`

Type: Boolean

Required: No

Specifies whether a volume is configurable at launch. When set to `true`, you can configure the volume when running a standalone task, or when creating or updating a service. When set to `true`, you won't be able to provide another volume configuration in the task definition. This parameter must be set to `true` to configure an Amazon EBS volume for attachment to a task. Setting `configuredAtLaunch` to `true` and deferring volume configuration to the launch phase allows you to create task definitions that aren't constrained to a volume type or to specific volume settings. Doing this makes your task definition reusable across different execution environments. For more information, see Amazon EBS volumes.

`dockerVolumeConfiguration`

Type: DockerVolumeConfiguration Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are supported only when running tasks on EC2 instances. Windows containers support only the use of the `local` driver. To use bind mounts, specify a `host` instead.

`scope`

Type: String

Valid Values: `task` | `shared`

Required: No

The scope for the Docker volume, which determines its lifecycle. Docker volumes that are scoped to a `task` are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as `shared` persist after the task stops.

`autoprovision`

Type: Boolean

Default value: `false`

Required: No

If this value is `true`, the Docker volume is created if it doesn't already exist. This field is used only if the `scope` is `shared`. If the `scope` is `task`, then this parameter must be omitted.

`driver`

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because this name is used for task placement. If the driver was installed by using the Docker plugin CLI, use `docker plugin ls` to retrieve the driver name from your container instance. If the driver was installed by using another method, use Docker plugin discovery to retrieve the driver name.

`driverOpts`

Type: String

Required: No

A map of Docker driver-specific options to pass through. This parameter maps to `DriverOpts` in the Create a volume section of Docker.

`labels`

Type: String

Required: No

Custom metadata to add to your Docker volume.

`efsVolumeConfiguration`

Type: EFSVolumeConfiguration Object

Required: No

This parameter is specified when using Amazon EFS volumes.

`fileSystemId`

Type: String

Required: Yes

The Amazon EFS file system ID to use.

`rootDirectory`

Type: String

Required: No

The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume will be used. Specifying / has the same effect as omitting this parameter.

> **Important**
>
> If an EFS access point is specified in the `authorizationConfig`, the root directory parameter must either be omitted or set to `/`, which will enforce the path set on the EFS access point.

`transitEncryption`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. If Amazon EFS IAM authorization is used, transit encryption must be enabled. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see Encrypting Data in Transit in the *Amazon Elastic File System User Guide*.

`transitEncryptionPort`

Type: Integer

Required: No

The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you don't specify a transit encryption port, the task will use the port selection strategy that the Amazon EFS mount helper uses. For more information, see EFS Mount Helper in the *Amazon Elastic File System User Guide*.

`authorizationConfig`

Type: EFSAuthorizationConfig Object

Required: No

The authorization configuration details for the Amazon EFS file system.

`accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value in the `efsVolumeConfiguration` must either be omitted or set to `/`, which will enforce the path set on the EFS access point. If an access point is used, transit encryption must be enabled in the `EFSVolumeConfiguration`. For more information, see Working with Amazon EFS Access Points in the *Amazon Elastic File System User Guide*.

`iam`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to use the Amazon ECS task IAM role that's defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the `EFSVolumeConfiguration`. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see IAM Roles for Tasks.

`FSxWindowsFileServerVolumeConfiguration`

Type: FSxWindowsFileServerVolumeConfiguration Object

Required: Yes

This parameter is specified when you're using an Amazon FSx for Windows File Server file system for task storage.

`fileSystemId`

Type: String

Required: Yes

The FSx for Windows File Server file system ID to use.

`rootDirectory`

Type: String

Required: Yes

The directory within the FSx for Windows File Server file system to mount as the root directory inside the host.

`authorizationConfig`

`credentialsParameter`

Type: String

Required: Yes

The authorization credential options.

options:

- Amazon Resource Name (ARN) of an AWS Secrets Manager secret.
- ARN of an AWS Systems Manager parameter.

`domain`

Type: String

Required: Yes

A fully qualified domain name hosted by an AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) directory or a self-hosted EC2 Active Directory.

Tags

When you register a task definition, you can optionally specify metadata tags that are applied to the task definition. Tags help you categorize and organize your task definition. Each tag consists of a key and an optional value. You define both of them. For more information, see Tagging Amazon ECS resources.

> **Important**
>
> Don't add personally identifiable information or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags aren't intended to be used for private or sensitive data.

The following parameters are allowed in a tag object.

`key`

Type: String

Required: No

One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

`value`

Type: String

Required: No

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

Other task definition parameters

The following task definition parameters can be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see Creating an Amazon ECS task definition using the console.

Topics

Ephemeral storage

`ephemeralStorage`

Type: EphemeralStorage object

Required: No

The amount of ephemeral storage (in GB) to allocate for the task. This parameter is used to expand the total amount of ephemeral storage available, beyond the default amount, for tasks that are hosted on AWS Fargate. For more information, see the section called "Bind mounts".

> **Note**
>
> This parameter is only supported on platform version `1.4.0` or later (Linux) or `1.0.0` or later (Windows).

IPC mode

`ipcMode`

This is not supported for tasks running on Fargate.

Type: String

Required: No

The IPC resource namespace to use for the containers in the task. The valid values are `host`, `task`, or `none`. If `host` is specified, then all the containers that are within the tasks that specified the `host` IPC mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If `task` is specified, all the containers that are within the specified task share the same IPC resources. If `none` is specified, then IPC resources within the containers of a task are private and not shared with other containers in a task or on the container instance. If no value is specified, then the IPC resource namespace sharing depends on the Docker daemon setting on the container instance.

If the `host` IPC mode is used, there's a heightened risk of undesired IPC namespace exposure.

If you're setting namespaced kernel parameters that use `systemControls` for the containers in the task, the following applies to your IPC resource namespace.

- For tasks that use the `host` IPC mode, IPC namespace that's related `systemControls` aren't supported.
- For tasks that use the `task` IPC mode, `systemControls` that relate to the IPC namespace apply to all containers within a task.

> **Note**
>
> This parameter is not supported for Windows containers or tasks using the Fargate launch type.

PID mode

`pidMode`

Type: String

Valid Values: `host` | `task`

Required: No

The process namespace to use for the containers in the task. The valid values are `host` or `task`. On Linux containers, the only valid value is `task`. For example, monitoring sidecars might need `pidMode` to access information about other containers running in the same task.

If `task` is specified, all containers within the specified task share the same process namespace.

If no value is specified, the default is a private namespace for each container.

> **Note**
>
> This parameter is only supported for tasks that are hosted on AWS Fargate if the tasks are using platform version `1.4.0` or later (Linux). This isn't supported for Windows containers on Fargate.

Fault injection

`enableFaultInjection`

Type: Boolean

Valid Values: `true` | `false`

Required: No

If this parameter is set to `true`, in a task's payload, Amazon ECS and Fargate accept fault injection requests from the task's containers. By default, this parameter is set to `false`.

Amazon ECS task definition parameters for the EC2 launch type

Task definitions are split into separate parts: the task family, the AWS Identity and Access Management (IAM) task role, the network mode, container definitions, volumes, task placement constraints, and launch types. The family and container definitions are required in a task definition. In contrast, task role, network mode, volumes, task placement constraints, and launch type are optional.

You can use these parameters in a JSON file to configure your task definition.

The following are more detailed descriptions for each task definition parameter for the EC2 launch type.

Family

`family`

Type: String

Required: Yes

When you register a task definition, you give it a family, which is similar to a name for multiple versions of the task definition, specified with a revision number. The first task definition that's registered into a particular family is given a revision of 1, and any task definitions registered after that are given a sequential revision number.

Launch types

When you register a task definition, you can specify a launch type that Amazon ECS should validate the task definition against. If the task definition doesn't validate against the compatibilities specified, a client exception is returned. For more information, see [Amazon ECS launch types](#).

The following parameter is allowed in a task definition.

`requiresCompatibilities`

Type: String array

Required: No

Valid Values: EC2

The launch type to validate the task definition against. This initiates a check to ensure that all of the parameters that are used in the task definition meet the requirements of the launch type.

Task role

`taskRoleArn`

Type: String

Required: No

When you register a task definition, you can provide a task role for an IAM role that allows the containers in the task permission to call the AWS APIs that are specified in its associated policies on your behalf. For more information, see Amazon ECS task IAM role.

When you launch the Amazon ECS-optimized Windows Server AMI, IAM roles for tasks on Windows require that the `-EnableTaskIAMRole` option is set. Your containers must also run some configuration code to use the feature. For more information, see Amazon EC2 Windows instance additional configuration.

Task execution role

`executionRoleArn`

Type: String

Required: Conditional

The Amazon Resource Name (ARN) of the task execution role that grants the Amazon ECS container agent permission to make AWS API calls on your behalf.

> **Note**
>
> The task execution IAM role is required depending on the requirements of your task. For more information, see Amazon ECS task execution IAM role.

Network mode

`networkMode`

Type: String

Required: No

The Docker networking mode to use for the containers in the task. For Amazon ECS tasks that are hosted on Amazon EC2 Linux instances, the valid values are none, `bridge`, `awsvpc`, and `host`. If no network mode is specified, the default network mode is `bridge`. For Amazon ECS tasks hosted on Amazon EC2 Windows instances, the valid values are `default`, and `awsvpc`. If no network mode is specified, the `default` network mode is used.

If the network mode is set to none, the task's containers don't have external connectivity and port mappings can't be specified in the container definition.

If the network mode is `bridge`, the task uses Docker's built-in virtual network on Linux, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Linux uses the `bridge` Docker network driver.

If the network mode is `host`, the task uses the host's network which bypasses Docker's built-in virtual network by mapping container ports directly to the ENI of the Amazon EC2 instance that hosts the task. Dynamic port mappings can't be used in this network mode. A container in a task definition that uses this mode must specify a specific `hostPort` number. A port number on a host can't be used by multiple tasks. As a result, you can't run multiple tasks of the same task definition on a single Amazon EC2 instance.

> **Important**
>
> When running tasks that use the `host` network mode, do not run containers using the root user (UID 0) for better security. As a security best practice, always use a non-root user.

If the network mode is `awsvpc`, the task is allocated an elastic network interface, and you must specify a `NetworkConfiguration` when you create a service or run a task with the task definition. For more information, see [Amazon ECS task networking options for the EC2 launch type](#).

If the network mode is `default`, the task uses Docker's built-in virtual network on Windows, which runs inside each Amazon EC2 instance that hosts the task. The built-in virtual network on Windows uses the `nat` Docker network driver.

The `host` and `awsvpc` network modes offer the highest networking performance for containers because they use the Amazon EC2 network stack. With the `host` and `awsvpc` network modes, exposed container ports are mapped directly to the corresponding host port (for the `host` network mode) or the attached elastic network interface port (for the `awsvpc` network mode). Because of this, you can't use dynamic host port mappings.

The allowable network mode depends on the underlying EC2 instance's operating system. If Linux, any network mode can be used. If Windows, the `default`, and `awsvpc` modes can be used.

Runtime platform

`operatingSystemFamily`

Type: String

Required: Conditional

Default: LINUX

When you register a task definition, you specify the operating system family.

The valid values are `LINUX`, `WINDOWS_SERVER_2025_FULL`, `WINDOWS_SERVER_2025_CORE`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`, `WINDOWS_SERVER_2016_FULL`, `WINDOWS_SERVER_2004_CORE`, and `WINDOWS_SERVER_20H2_CORE`.

All task definitions that are used in a service must have the same value for this parameter.

When a task definition is part of a service, this value must match the service `platformFamily` value.

`cpuArchitecture`

Type: String

Required: Conditional

Default: X86_64

When you register a task definition, you specify the CPU architecture. The valid values are `X86_64` and `ARM64`.

All task definitions that are used in a service must have the same value for this parameter.

When you have Linux tasks, you can set the value to `ARM64`. For more information, see [the section called "Task definitions for 64-bit ARM workloads"](#).

Task size

When you register a task definition, you can specify the total CPU and memory used for the task. This is separate from the `cpu` and `memory` values at the container definition level. For tasks that are hosted on Amazon EC2 instances, these fields are optional.

> **Note**
>
> Task-level CPU and memory parameters are ignored for Windows containers. We recommend specifying container-level resources for Windows containers.

`cpu`

Type: String

Required: Conditional

> **Note**
>
> This parameter is not supported for Windows containers.

The hard limit of CPU units to present for the task. You can specify CPU values in the JSON file as a string in CPU units or virtual CPUs (vCPUs). For example, you can specify a CPU value either as `1024` in CPU units or `1 vCPU` in vCPUs. When the task definition is registered, a vCPU value is converted to an integer indicating the CPU units.

This field is optional. If your cluster doesn't have any registered container instances with the requested CPU units available, the task fails. Supported values are between `0.125` vCPUs and `192` vCPUs.

`memory`

Type: String

Required: Conditional

> **Note**
>
> This parameter is not supported for Windows containers.

The hard limit of memory to present to the task. You can specify memory values in the task definition as a string in mebibytes (MiB) or gigabytes (GB). For example, you can specify a memory value either as `3072` in MiB or `3 GB`in GB. When the task definition is registered, a GB value is converted to an integer indicating the MiB.

This field is optional and any value can be used. If a task-level memory value is specified, then the container-level memory value is optional. If your cluster doesn't have any registered container instances with the requested memory available, the task fails. You can maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type. For more information, see Reserving Amazon ECS Linux container instance memory.

Container definitions

When you register a task definition, you must specify a list of container definitions that are passed to the Docker daemon on a container instance. The following parameters are allowed in a container definition.

Topics

- Standard container definition parameters
- Advanced container definition parameters
- Other container definition parameters

Standard container definition parameters

The following task definition parameters are either required or used in most container definitions.

Topics

- Name

- Image
- Memory
- Port mappings
- Private Repository Credentials

Name

`name`

Type: String

Required: Yes

The name of a container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. If you're linking multiple containers in a task definition, the `name` of one container can be entered in the `links` of another container. This is to connect the containers.

Image

`image`

Type: String

Required: Yes

The image used to start a container. This string is passed directly to the Docker daemon. By default, images in the Docker Hub registry are available. You can also specify other repositories with either `*repository-url*/*image*:*tag*` or `*repository-url*/*image*@*digest*`. Up to 255 letters (uppercase and lowercase), numbers, hyphens, underscores, colons, periods, forward slashes, and number signs are allowed. This parameter maps to `Image` in the docker create-container command and the `IMAGE` parameter of the docker run command.

- When a new task starts, the Amazon ECS container agent pulls the latest version of the specified image and tag for the container to use. However, subsequent updates to a repository image aren't propagated to already running tasks.
- Whenyou don't specify a tag or digest in the image path in the task definition, the Amazon ECS container agent pulls the latest version of the specified image.

- However, subsequent updates to a repository image aren't propagated to already running tasks.
- Images in private registries are supported. For more information, see [Using non-AWS container images in Amazon ECS]().
- Images in Amazon ECR repositories can be specified by using either the full `registry/repository:tag` or `registry/repository@digest` naming convention (for example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest` or `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app@sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE`).
- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

`versionConsistency`

Type: String

Valid values: `enabled|disabled`

Required: No

Specifies whether Amazon ECS will resolve the container image tag provided in the container definition to an image digest. By default, this behavior is `enabled`. If you set the value for a container as `disabled`, Amazon ECS will not resolve the container image tag to a digest and will use the original image URI specified in the container definition for deployment. For more information about container image resolution, see [Container image resolution]().

Memory

`memory`

Type: Integer

Required: No

The amount (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. The total amount of memory reserved for all containers within a task must be lower than the task `memory` value, if one is specified. This parameter maps to `Memory` in the docker create-container command and the `--memory` option to docker run.

You must specify either a task-level memory value or a container-level memory value. If you specify both a container-level `memory` and `memoryReservation` value, the `memory` value must be greater than the `memoryReservation` value. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance that the container is placed on. Otherwise, the value of `memory` is used.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

> **Note**
>
> If you're trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Reserving Amazon ECS Linux container instance memory]().

`memoryReservation`

Type: Integer

Required: No

The soft limit (in MiB) of memory to reserve for the container. When system memory is under contention, Docker attempts to keep the container memory to this soft limit. However, your container can use more memory when needed. The container can use up to the hard limit that's specified with the `memory` parameter (if applicable) or all of the available memory on the container instance, whichever comes first. This parameter maps to `MemoryReservation` in the docker create-container command and the `--memory-reservation` option to docker run.

If a task-level memory value isn't specified, you must specify a non-zero integer for one or both of `memory` or `memoryReservation` in a container definition. If you specify both, `memory` must be greater than `memoryReservation`. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance that the container is placed on. Otherwise, the value of `memory` is used.

For example, suppose that your container normally uses 128 MiB of memory, but occasionally bursts to 256 MiB of memory for short periods of time. You can set a `memoryReservation` of 128 MiB, and a `memory` hard limit of 300 MiB. This configuration allows the container to only reserve 128 MiB of memory from the remaining resources on the container instance. At the same time, this configuration also allows the container to use more memory resources when needed.

> **Note**
>
> This parameter isn't supported for Windows containers.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

> **Note**
>
> If you're trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Reserving Amazon ECS Linux container instance memory](#).

Port mappings

`portMappings`

Type: Object array

Required: No

Port mappings expose your container's network ports to the outside world. this allows clients to access your application. It's also used for inter-container communication within the same task.

For task definitions that use the `awsvpc` network mode, only specify the `containerPort`. The `hostPort` is always ignored, and the container port is automatically mapped to a random high-numbered port on the host.

Port mappings on Windows use the `NetNAT` gateway address rather than `localhost`. There's no loopback for port mappings on Windows, so you can't access a container's mapped port from the host itself.

Most fields of this parameter (including `containerPort`, `hostPort`, `protocol`) map to `PortBindings` in thedocker create-container command and the `--publish` option to docker run. If the network mode of a task definition is set to `host`, host ports must either be undefined or match the container port in the port mapping.

> **Note**
>
> After a task reaches the `RUNNING` status, manual and automatic host and container port assignments are visible in the following locations:
>
> - Console: The **Network Bindings** section of a container description for a selected task.
> - AWS CLI: The `networkBindings` section of the **describe-tasks** command output.
> - API: The `DescribeTasks` response.
> - Metadata: The task metadata endpoint.

`appProtocol`

Type: String

Required: No

The application protocol that's used for the port mapping. This parameter only applies to Service Connect. We recommend that you set this parameter to be consistent with the protocol that your application uses. If you set this parameter, Amazon ECS adds protocol-specific connection handling to the service connect proxy. If you set this parameter, Amazon ECS adds protocol-specific telemetry in the Amazon ECS console and CloudWatch.

If you don't set a value for this parameter, then TCP is used. However, Amazon ECS doesn't add protocol-specific telemetry for TCP.

For more information, see [the section called “ Service Connect”].

Valid protocol values: `"http"` | `"http2"` | `"grpc"`

`containerPort`

Type: Integer

Required: Yes, when `portMappings` are used

The port number on the container that's bound to the user-specified or automatically assigned host port.

For tasks that use the `awsvpc` network mode, you use `containerPort` to specify the exposed ports.

Suppose that you're using containers in a task with the EC2 launch type and you specify a container port and not a host port. Then, your container automatically receives a host port in the ephemeral port range. For more information, see `hostPort`. Port mappings that are automatically assigned in this way don't count toward the 100 reserved ports quota of a container instance.

`containerPortRange`

Type: String

Required: No

The port number range on the container that's bound to the dynamically mapped host port range.

You can only set this parameter by using the `register-task-definition` API. The option is available in the `portMappings` parameter. For more information, see register-task-definition in the *AWS Command Line Interface Reference*.

The following rules apply when you specify a `containerPortRange`:

- You must use either the `bridge` network mode or the `awsvpc` network mode.
- This parameter is available for both the Linux and Windows operating systems.
- The container instance must have at least version 1.67.0 of the container agent and at least version 1.67.0-1 of the `ecs-init` package.
- You can specify a maximum of 100 port ranges for each container.

- You don't specify a `hostPortRange`. The value of the `hostPortRange` is set as follows:
  - For containers in a task with the `awsvpc` network mode, the `hostPort` is set to the same value as the `containerPort`. This is a static mapping strategy.
  - For containers in a task with the `bridge` network mode, the Amazon ECS agent finds open host ports from the default ephemeral range and passes it to docker to bind them to the container ports.
- The `containerPortRange` valid values are between 1 and 65535.
- A port can only be included in one port mapping for each container.
- You can't specify overlapping port ranges.
- The first port in the range must be less than last port in the range.
- Docker recommends that you turn off the docker-proxy in the Docker daemon config file when you have a large number of ports.

  For more information, see [Issue #11185] on GitHub.

  For information about how to turn off the docker-proxy in the Docker daemon config file, see [Docker daemon] in the *Amazon ECS Developer Guide*.

You can call [DescribeTasks] to view the `hostPortRange`, which are the host ports that are bound to the container ports.

The port ranges aren't included in the Amazon ECS task events, which are sent to EventBridge. For more information, see [the section called “Automate responses to Amazon ECS errors using EventBridge”].

`hostPortRange`

Type: String

Required: No

The port number range on the host that's used with the network binding. This is assigned by Docker and delivered by the Amazon ECS agent.

`hostPort`

Type: Integer

Required: No

The port number on the container instance to reserve for your container.

You can specify a non-reserved host port for your container port mapping. This is referred to as *static* host port mapping. Or, you can omit the `hostPort` (or set it to `0`) while specifying a `containerPort`. Your container automatically receives a port in the ephemeral port range for your container instance operating system and Docker version. This is referred to as *dynamic* host port mapping.

The default ephemeral port range Docker version 1.6.0 and later is listed on the instance under `/proc/sys/net/ipv4/ip_local_port_range`. If this kernel parameter is unavailable, the default ephemeral port range from `49153–65535` is used. Don't attempt to specify a host port in the ephemeral port range. This is because these are reserved for automatic assignment. In general, ports under `32768` are outside of the ephemeral port range.

The default reserved ports are `22` for SSH, the Docker ports `2375` and `2376`, and the Amazon ECS container agent ports `51678-51680`. Any host port that was previously user-specified for a running task is also reserved while the task is running. After a task stops, the host port is released. The current reserved ports are displayed in the `remainingResources` of the **describe-container-instances** output. A container instance might have up to 100 reserved ports at a time, including the default reserved ports. Automatically assigned ports don't count toward the 100 reserved ports quota.

`name`

Type: String

Required: No, required for Service Connect and VPC Lattice to be configured in a service

The name that's used for the port mapping. This parameter only applies to Service Connect and VPC Lattice. This parameter is the name that you use in the Service Connect and VPC Lattice configuration of a service.

For more information, see [Use Service Connect to connect Amazon ECS services with short names].

In the following example, both of the required fields for Service Connect and VPC Lattice are used.

```
"portMappings": [
    {
        "name": string,
        "containerPort": integer
    }
]
```

`protocol`

Type: String

Required: No

The protocol that's used for the port mapping. Valid values are `tcp` and `udp`. The default is `tcp`.

> **Important**
>
> Only `tcp` is supported for Service Connect. Remember that `tcp` is implied if this field isn't set.

> **Important**
>
> UDP support is only available on container instances that were launched with version 1.2.0 of the Amazon ECS container agent (such as the `amzn-ami-2015.03.c-amazon-ecs-optimized` AMI) or later, or with container agents that have been updated to version 1.3.0 or later. To update your container agent to the latest version, see [Updating the Amazon ECS container agent](#).

If you're specifying a host port, use the following syntax.

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
```

```
]
```

If you want an automatically assigned host port, use the following syntax.

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
]
```

Private Repository Credentials

`repositoryCredentials`

Type: RepositoryCredentials object

Required: No

The repository credentials for private registry authentication.

For more information, see Using non-AWS container images in Amazon ECS.

`credentialsParameter`

Type: String

Required: Yes, when `repositoryCredentials` are used

The Amazon Resource Name (ARN) of the secret containing the private repository credentials.

For more information, see Using non-AWS container images in Amazon ECS.

> **Note**
>
> When you use the Amazon ECS API, AWS CLI, or AWS SDKs, if the secret exists in the same Region as the task that you're launching then you can use either the full ARN or the name of the secret. When you use the AWS Management Console, you must specify the full ARN of the secret.

The following is a snippet of a task definition that shows the required parameters:

```
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
            "credentialsParameter":
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
        }
    }
]
```

Advanced container definition parameters

The following advanced container definition parameters provide extended capabilities to the docker run command that's used to launch containers on your Amazon ECS container instances.

Topics

- Restart policy
- Health check
- Environment
- Network settings
- Storage and logging
- Security
- Resource limits
- Docker labels

Restart policy

`restartPolicy`

The container restart policy and associated configuration parameters. When you set up a restart policy for a container, Amazon ECS can restart the container without needing to replace the task. For more information, see Restart individual containers in Amazon ECS tasks with container restart policies.

`enabled`

Type: Boolean

Required: Yes

Specifies whether a restart policy is enabled for the container.

`ignoredExitCodes`

Type: Integer array

Required: No

A list of exit codes that Amazon ECS will ignore and not attempt a restart on. You can specify a maximum of 50 container exit codes. By default, Amazon ECS does not ignore any exit codes.

`restartAttemptPeriod`

Type: Integer

Required: No

A period of time (in seconds) that the container must run for before a restart can be attempted. A container can be restarted only once every `restartAttemptPeriod` seconds. If a container isn't able to run for this time period and exits early, it will not be restarted. You can set a minimum `restartAttemptPeriod` of 60 seconds and a maximum `restartAttemptPeriod` of 1800 seconds. By default, a container must run for 300 seconds before it can be restarted.

Health check

`healthCheck`

The container health check command and the associated configuration parameters for the container. For more information, see Determine Amazon ECS task health using container health checks.

`command`

A string array that represents the command that the container runs to determine if it's healthy. The string array can start with `CMD` to run the command arguments directly, or `CMD-`

SHELL to run the command with the container's default shell. If neither is specified, CMD is used.

When registering a task definition in the AWS Management Console, use a comma separated list of commands. These commands are converted to a string after the task definition is created. An example input for a health check is the following.

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

When registering a task definition using the AWS Management Console JSON panel, the AWS CLI, or the APIs, enclose the list of commands in brackets. An example input for a health check is the following.

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

An exit code of 0, with no `stderr` output, indicates success, and a non-zero exit code indicates failure.

`interval`

The period of time (in seconds) between each health check. You can specify between 5 and 300 seconds. The default value is 30 seconds.

`timeout`

The period of time (in seconds) to wait for a health check to succeed before it's considered a failure. You can specify between 2 and 60 seconds. The default value is 5 seconds.

`retries`

The number of times to retry a failed health check before the container is considered unhealthy. You can specify between 1 and 10 retries. The default value is three retries.

`startPeriod`

The optional grace period to provide containers time to bootstrap in before failed health checks count towards the maximum number of retries. You can specify between 0 and 300 seconds. By default, `startPeriod` is disabled.

If a health check succeeds within the `startPeriod`, then the container is considered healthy and any subsequent failures count toward the maximum number of retries.

Environment

`cpu`

Type: Integer

Required: No

The number of `cpu` units the Amazon ECS container agent reserves for the container. On Linux, this parameter maps to `CpuShares` in the [Create a container](#) section.

> **Note**
>
> You can determine the number of CPU units that are available to each Amazon EC2 instance type. To do this, multiply the number of vCPUs listed for that instance type on the [Amazon EC2 Instances](#) detail page by 1,024.

Linux containers share unallocated CPU units with other containers on the container instance with the same ratio as their allocated amount. For example, assume that you run a single-container task on a single-core instance type with 512 CPU units specified for that container. Moreover, that task is the only task running on the container instance. In this example, the container can use the full 1,024 CPU unit share at any given time. However, assume then that you launched another copy of the same task on that container instance. Each task is guaranteed a minimum of 512 CPU units when needed. Similarly, if the other container isn't using the remaining CPU, each container can float to higher CPU usage. However, if both tasks were 100% active all of the time, they are limited to 512 CPU units.

On Linux container instances, the Docker daemon on the container instance uses the CPU value to calculate the relative CPU share ratios for running containers. The minimum valid CPU share value that the Linux kernel allows is 2, and the maximum valid CPU share value that the Linux kernel allows is 262144. However, the CPU parameter isn't required, and you can use CPU values below two and above 262144 in your container definitions. For CPU values below two (including null) and above 262144, the behavior varies based on your Amazon ECS container agent version:

- **Agent versions <= 1.1.0:** Null and zero CPU values are passed to Docker as 0. Docker then converts this value to 1,024 CPU shares. CPU values of one are passed to Docker as one, which the Linux kernel converts to two CPU shares.

- **Agent versions >= 1.2.0:** Null, zero, and CPU values of one are passed to Docker as two CPU shares.
- **Agent versions >= 1.84.0:** CPU values greater than 256 vCPU are passed to Docker as 256, which is equivalent to 262144 CPU shares.

On Windows container instances, the CPU quota is enforced as an absolute quota. Windows containers only have access to the specified amount of CPU that's defined in the task definition. A null or zero CPU value is passed to Docker as `0`. Windows then interprets this value as 1% of one CPU.

For more examples, see How Amazon ECS manages CPU and memory resources.

`gpu`

Type: ResourceRequirement object

Required: No

The number of physical `GPUs` that the Amazon ECS container agent reserves for the container. The number of GPUs reserved for all containers in a task must not exceed the number of available GPUs on the container instance the task is launched on. For more information, see Amazon ECS task definitions for GPU workloads.

> **Note**
>
> This parameter isn't supported for Windows containers.

`Elastic Inference accelerator`

Type: ResourceRequirement object

Required: No

For the `InferenceAccelerator` type, the `value` matches the `deviceName` for an `InferenceAccelerator` specified in a task definition. For more information, see the section called “Elastic Inference accelerator name”.

> **Note**
>
> This parameter isn't supported for Windows containers.

`essential`

Type: Boolean

Required: No

Suppose that the `essential` parameter of a container is marked as `true`, and that container fails or stops for any reason. Then, all other containers that are part of the task are stopped. If the `essential` parameter of a container is marked as `false`, then its failure doesn't affect the rest of the containers in a task. If this parameter is omitted, a container is assumed to be essential.

All tasks must have at least one essential container. Suppose that you have an application that's composed of multiple containers. Then, group containers that are used for a common purpose into components, and separate the different components into multiple task definitions. For more information, see Architect your application for Amazon ECS.

```
"essential": true|false
```

`entryPoint`

> **Important**
>
> Early versions of the Amazon ECS container agent don't properly handle `entryPoint` parameters. If you have problems using `entryPoint`, update your container agent or enter your commands and arguments as `command` array items instead.

Type: String array

Required: No

The entry point that's passed to the container.

```
"entryPoint": ["string", ...]
```

`command`

Type: String array

Required: No

The command that's passed to the container. This parameter maps to `Cmd` in the create-container command and the `COMMAND` parameter to docker run. If there are multiple arguments, make sure that each argument is a separated string in the array.

```
"command": ["string", ...]
```

`workingDirectory`

Type: String

Required: No

The working directory to run commands inside the container in. This parameter maps to `WorkingDir` in the Create a container section of the Docker Remote API and the `--workdir` option to **docker run**.

```
"workingDirectory": "string"
```

`environmentFiles`

Type: Object array

Required: No

A list of files containing the environment variables to pass to a container. This parameter maps to the `--env-file` option to the docker run command.

When FIPS is enabled, bucket names that have periods (.) (for example, amzn-s3-demo-bucket1.name.example) aren't supported. Having periods (.) in the bucket name prevents the task from starting because the agent can't pull the environment variable file from Amazon S3.

This isn't available for Windows containers.

You can specify up to 10 environment files. The file must have a `.env` file extension. Each line in an environment file contains an environment variable in `VARIABLE=VALUE` format. Lines that start with # are treated as comments and are ignored.

If there are individual environment variables specified in the container definition, they take precedence over the variables contained within an environment file. If multiple environment files are specified that contain the same variable, they're processed from the top down. We recommend that you use unique variable names. For more information, see Pass an individual environment variable to an Amazon ECS container.

`value`

Type: String

Required: Yes

The Amazon Resource Name (ARN) of the Amazon S3 object containing the environment variable file.

`type`

Type: String

Required: Yes

The file type to use. The only supported value is `s3`.

`environment`

Type: Object array

Required: No

The environment variables to pass to a container. This parameter maps to `Env` in the docker create-container command and the `--env` option to the docker run command.

> **Important**
> We do not recommend using plaintext environment variables for sensitive information, such as credential data.

`name`

Type: String

Required: Yes, when `environment` is used

The name of the environment variable.

`value`

Type: String

Required: Yes, when `environment` is used

The value of the environment variable.

```
"environment" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

`secrets`

Type: Object array

Required: No

An object that represents the secret to expose to your container. For more information, see Pass sensitive data to an Amazon ECS container.

`name`

Type: String

Required: Yes

The value to set as the environment variable on the container.

`valueFrom`

Type: String

Required: Yes

The secret to expose to the container. The supported values are either the full Amazon Resource Name (ARN) of the AWS Secrets Manager secret or the full ARN of the parameter in the AWS Systems Manager Parameter Store.

> **Note**
>
> If the Systems Manager Parameter Store parameter or Secrets Manager parameter exists in the same AWS Region as the task that you're launching, you can use either

the full ARN or name of the secret. If the parameter exists in a different Region, then the full ARN must be specified.

```
"secrets": [
    {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }
]
```

Network settings

`disableNetworking`

Type: Boolean

Required: No

When this parameter is true, networking is off within the container.

> **Note**
>
> This parameter isn't supported for Windows containers or tasks using the `awsvpc` network mode.

The default is `false`.

```
"disableNetworking": true|false
```

`links`

Type: String array

Required: No

The `link` parameter allows containers to communicate with each other without the need for port mappings. This parameter is only supported if the network mode of a task definition is set to `bridge`. The `name:internalName` construct is analogous to `name:alias` in Docker links. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed..

> **Note**
>
> This parameter isn't supported for Windows containers or tasks using the `awsvpc` network mode.

> **Important**
>
> Containers that are collocated on the same container instance might communicate with each other without requiring links or host port mappings. The network isolation on a container instance is controlled by security groups and VPC settings.

```
"links": ["name:internalName", ...]
```

`hostname`

Type: String

Required: No

The hostname to use for your container. This parameter maps to `Hostname` in the docker create-container and the `--hostname` option to docker run.

> **Note**
>
> If you're using the `awsvpc` network mode, the `hostname` parameter isn't supported.

```
"hostname": "string"
```

`dnsServers`

Type: String array

Required: No

A list of DNS servers that are presented to the container.

> **Note**
>
> This parameter isn't supported for Windows containers or tasks using the `awsvpc` network mode.

```
"dnsServers": ["string", ...]
```

`dnsSearchDomains`

Type: String array

Required: No

Pattern: ^[a-zA-Z0-9-.]{0,253}[a-zA-Z0-9]$

A list of DNS search domains that are presented to the container. This parameter maps to `DnsSearch` in the docker create-container command the `--dns-search` option to docker run.

> **Note**
>
> This parameter isn't supported for Windows containers or tasks that use the `awsvpc` network mode.

```
"dnsSearchDomains": ["string", ...]
```

`extraHosts`

Type: Object array

Required: No

A list of hostnames and IP address mappings to append to the `/etc/hosts` file on the container.

This parameter maps to `ExtraHosts` in the docker create-container command and the `--add-host` option to docker run.

> **Note**
>
> This parameter isn't supported for Windows containers or tasks that use the `awsvpc` network mode.

```
"extraHosts": [
      {
        "hostname": "string",
        "ipAddress": "string"
      }
      ...
    ]
```

`hostname`

Type: String

Required: Yes, when `extraHosts` are used

The hostname to use in the `/etc/hosts` entry.

`ipAddress`

Type: String

Required: Yes, when `extraHosts` are used

The IP address to use in the `/etc/hosts` entry.

Storage and logging

`readonlyRootFilesystem`

Type: Boolean

Required: No

When this parameter is true, the container is given read-only access to its root file system. This parameter maps to `ReadonlyRootfs` in the docker create-container command the `--read-only` option to docker run.

> **Note**
>
> This parameter is not supported for Windows containers.

The default is `false`.

```
"readonlyRootFilesystem": true|false
```

`mountPoints`

Type: Object array

Required: No

The mount points for the data volumes in your container. This parameter maps to `Volumes` in the create-container Docker API and the `--volume` option to docker run.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount points cannot be used across drives. You must specify mount points to attach an Amazon EBS volume directly to an Amazon ECS task.

`sourceVolume`

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

`containerPath`

Type: String

Required: Yes, when `mountPoints` are used

The path in the container where the volume will be mounted.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

For tasks that run the Windows operating system, leave the value as the default of `false`.

`volumesFrom`

Type: Object array

Required: No

Data volumes to mount from another container. This parameter maps to `VolumesFrom` in the docker create-container command and the `--volumes-from` option to docker run.

`sourceContainer`

Type: String

Required: Yes, when `volumesFrom` is used

The name of the container to mount volumes from.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

```
"volumesFrom": [
                {
                  "sourceContainer": "string",
                  "readOnly": true|false
                }
              ]
```

`logConfiguration`

Type: LogConfiguration Object

Required: No

The log configuration specification for the container.

For example task definitions that use a log configuration, see Example Amazon ECS task definitions.

This parameter maps to `LogConfig` in the docker create-container command and the `--log-driver` option to docker run. By default, containers use the same logging driver that the Docker daemon uses. However, the container might use a different logging driver than the Docker daemon by specifying a log driver with this parameter in the container definition. To use a different logging driver for a container, the log system must be configured properly on the container instance (or on a different log server for remote logging options).

Consider the following when specifying a log configuration for your containers:

- Amazon ECS supports a subset of the logging drivers that are available to the Docker daemon. Additional log drivers might be available in future releases of the Amazon ECS container agent.
- This parameter requires version 1.18 or later of the Docker Remote API on your container instance.
- The Amazon ECS container agent that runs on a container instance must register the logging drivers that are available on that instance with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable before containers that are placed on that instance can use these log configuration options. For more information, see Amazon ECS container agent configuration.

```
"logConfiguration": {
      "logDriver": "awslogs","fluentd","gelf","json-
file","journald","splunk","syslog","awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
  "name": "string",
  "valueFrom": "string"
 }]
}
```

`logDriver`

Type: String

Valid values: `"awslogs","fluentd","gelf","json-file","journald","splunk","syslog","awsfirelens"`

Required: Yes, when `logConfiguration` is used

The log driver to use for the container. By default, the valid values that are listed earlier are log drivers that the Amazon ECS container agent can communicate with.

The supported log drivers are `awslogs`, `fluentd`, `gelf`, `json-file`, `journald`, `syslog`, `splunk`, and `awsfirelens`.

For more information about how to use the `awslogs` log driver in task definitions to send your container logs to CloudWatch Logs, see Send Amazon ECS logs to CloudWatch .

For more information about using the `awsfirelens` log driver, see Custom Log Routing.

> **Note**
>
> If you have a custom driver that isn't listed, you can fork the Amazon ECS container agent project that's available on GitHub and customize it to work with that driver. We encourage you to submit pull requests for changes that you want to have included. However, we don't currently support running modified copies of this software.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

`options`

Type: String to string map

Required: No

The key/value map of configuration options to send to the log driver.

The options you can specify depend on the log driver. Some of the options you can specify when you use the `awslogs` router to route logs to Amazon CloudWatch include the following:

`awslogs-create-group`

Required: No

Specify whether you want the log group to be created automatically. If this option isn't specified, it defaults to `false`.

> **Note**
>
> Your IAM policy must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`.

`awslogs-region`

Required: Yes

Specify the AWS Region that the `awslogs` log driver is to send your Docker logs to. You can choose to send all of your logs from clusters in different Regions to a single region in CloudWatch Logs. This is so that they're all visible in one location. Otherwise, you can separate them by Region for more granularity. Make sure that the specified log group exists in the Region that you specify with this option.

`awslogs-group`

Required: Yes

Make sure to specify a log group that the `awslogs` log driver sends its log streams to.

`awslogs-stream-prefix`

Required: Optional

Use the `awslogs-stream-prefix` option to associate a log stream with the specified prefix, the container name, and the ID of the Amazon ECS task that the container belongs to. If you specify a prefix with this option, then the log stream takes the following format.

```
prefix-name/container-name/ecs-task-id
```

If you don't specify a prefix with this option, then the log stream is named after the container ID that's assigned by the Docker daemon on the container instance. Because it's difficult to trace logs back to the container that sent them with just the Docker container ID (which is only available on the container instance), we recommend that you specify a prefix with this option.

For Amazon ECS services, you can use the service name as the prefix. Doing so, you can trace log streams to the service that the container belongs to, the name of the container that sent them, and the ID of the task that the container belongs to.

You must specify a stream-prefix for your logs to have your logs appear in the Log pane when using the Amazon ECS console.

`awslogs-datetime-format`

Required: No

This option defines a multiline start pattern in Python `strftime` format. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. The matched line is the delimiter between log messages.

One example of a use case for using this format is for parsing output such as a stack dump, which might otherwise be logged in multiple entries. The correct pattern allows it to be captured in a single entry.

For more information, see awslogs-datetime-format.

You cannot configure both the `awslogs-datetime-format` and `awslogs-multiline-pattern` options.

> **Note**
>
> Multiline logging performs regular expression parsing and matching of all log messages. This might have a negative impact on logging performance.

`awslogs-multiline-pattern`

Required: No

This option defines a multiline start pattern that uses a regular expression. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. The matched line is the delimiter between log messages.

For more information, see awslogs-multiline-pattern.

This option is ignored if `awslogs-datetime-format` is also configured.

You cannot configure both the `awslogs-datetime-format` and `awslogs-multiline-pattern` options.

> **Note**
>
> Multiline logging performs regular expression parsing and matching of all log messages. This might have a negative impact on logging performance.

The following options apply to all supported log drivers.

`mode`

Required: No

Valid values: `non-blocking` | `blocking`

This option defines the delivery mode of log messages from the container to the log driver specified using `logDriver`. The delivery mode you choose affects application availability when the flow of logs from the container is interrupted.

If you use the `blocking` mode and the flow of logs is interrupted, calls from container code to write to the `stdout` and `stderr` streams will block. The logging thread of the application will block as a result. This may cause the application to become unresponsive and lead to container healthcheck failure.

If you use the `non-blocking` mode, the container's logs are instead stored in an in-memory intermediate buffer configured with the `max-buffer-size` option. This prevents the application from becoming unresponsive when logs cannot be sent. We recommend using this mode if you want to ensure service availability and are okay with some log loss. For more information, see [Preventing log loss with non-blocking mode in the `awslogs` container log driver](#).

You can set a default `mode` for all containers in a specific AWS Region by using the `defaultLogDriverMode` account setting. If you don't specify the `mode` option in the `logConfiguration` or configure the account setting, Amazon ECS will default to `non-blocking` mode. For more information about the account setting, see [Default log driver mode](#).

When `non-blocking` mode is used, the `max-buffer-size` log option controls the size of the buffer that's used for intermediate message storage. Make sure to specify an adequate buffer size based on your application. The total amount of memory allocated at the task level should be greater than the amount of memory that's allocated for all the containers in addition to the log driver memory buffer.

> **Note**
>
> On June 25, 2025, Amazon ECS changed the default log driver mode from `blocking` to `non-blocking` to prioritize task availability over logging. To continue using the `blocking` mode after this change, do one of the following:
>
> - Set the `mode` option in your container definition's `logConfiguration` as `blocking`.
> - Set the `defaultLogDriverMode` account setting to `blocking`.

`max-buffer-size`

Required: No

Default value: `10 m`

When `non-blocking` mode is used, the `max-buffer-size` log option controls the size of the buffer that's used for intermediate message storage. Make sure to specify an adequate buffer size based on your application. When the buffer fills up, further logs cannot be stored. Logs that cannot be stored are lost.

To route logs using the `splunk` log router, you need to specify a `splunk-token` and a `splunk-url`.

When you use the `awsfirelens` log router to route logs to an AWS service or AWS Partner Network destination for log storage and analytics, you can set the `log-driver-buffer-limit` option to limit the number of events that are buffered in memory, before being sent to the log router container. It can help to resolve potential log loss issue because high throughput might result in memory running out for the buffer inside of Docker. For more information, see the section called “Configuring logs for high throughput”.

Other options you can specify when using `awsfirelens` to route logs depend on the destination. When you export logs to Amazon Data Firehose, you can specify the AWS Region with `region` and a name for the log stream with `delivery_stream`.

When you export logs to Amazon Kinesis Data Streams, you can specify an AWS Region with `region` and a data stream name with `stream`.

When you export logs to Amazon OpenSearch Service, you can specify options like `Name`, `Host` (OpenSearch Service endpoint without protocol), `Port`, `Index`, `Type`, `Aws_auth`, `Aws_region`, `Suppress_Type_Name`, and `tls`.

When you export logs to Amazon S3, you can specify the bucket using the `bucket` option. You can also specify `region`, `total_file_size`, `upload_timeout`, and `use_put_object` as options.

This parameter requires version 1.19 of the Docker Remote API or greater on your container instance.

`secretOptions`

Type: Object array

Required: No

An object that represents the secret to pass to the log configuration. Secrets that are used in log configuration can include an authentication token, certificate, or encryption key. For more information, see [Pass sensitive data to an Amazon ECS container](#).

`name`

Type: String

Required: Yes

The value to set as the environment variable on the container.

`valueFrom`

Type: String

Required: Yes

The secret to expose to the log configuration of the container.

```
"logConfiguration": {
 "logDriver": "splunk",
 "options": {
  "splunk-url": "https://cloud.splunk.com:8080",
  "splunk-token": "...",
  "tag": "...",
  ...
 },
```

```
  "secretOptions": [{
   "name": "splunk-token",
   "valueFrom": "/ecs/logconfig/splunkcred"
  }]
 }
```

`firelensConfiguration`

Type: FirelensConfiguration Object

Required: No

The FireLens configuration for the container. This is used to specify and configure a log router for container logs. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.

```
{
    "firelensConfiguration": {
        "type": "fluentd",
        "options": {
            "KeyName": ""
        }
    }
}
```

`options`

Type: String to string map

Required: No

The key/value map of options to use when configuring the log router. This field is optional and can be used to specify a custom configuration file or to add additional metadata, such as the task, task definition, cluster, and container instance details to the log event. If specified, the syntax to use is `"options":{"enable-ecs-log-metadata":"true|false","config-file-type:"s3|file","config-file-value":"arn:aws:s3:::amzn-s3-demo-bucket/fluent.conf|filepath"}`. For more information, see Example Amazon ECS task definition: Route logs to FireLens.

`type`

Type: String

Required: Yes

The log router to use. The valid values are `fluentd` or `fluentbit`.

Security

For more information about container security, see [Amazon ECS task and container security best practices](#).

`credentialSpecs`

Type: String array

Required: No

A list of ARNs in SSM or Amazon S3 to a credential spec (`CredSpec`) file that configures the container for Active Directory authentication. We recommend that you use this parameter instead of the `dockerSecurityOptions`. The maximum number of ARNs is 1.

There are two formats for each ARN.

credentialspecdomainless:MyARN

You use `credentialspecdomainless:MyARN` to provide a `CredSpec` with an additional section for a secret in Secrets Manager. You provide the login credentials to the domain in the secret.

Each task that runs on any container instance can join different domains.

You can use this format without joining the container instance to a domain.

credentialspec:MyARN

You use `credentialspec:MyARN` to provide a `CredSpec` for a single domain.

You must join the container instance to the domain before you start any tasks that use this task definition.

In both formats, replace `MyARN` with the ARN in SSM or Amazon S3.

The `credspec` must provide a ARN in Secrets Manager for a secret containing the username, password, and the domain to connect to. For better security, the instance isn't joined to the domain for domainless authentication. Other applications on the instance can't use the domainless credentials. You can use this parameter to run tasks on the same instance, even it the tasks need to join different domains. For more information, see [Using gMSAs for Windows Containers] and [Using gMSAs for Linux Containers].

`privileged`

Type: Boolean

Required: No

When this parameter is true, the container is given elevated privileges on the host container instance (similar to the `root` user). We recommend against running containers with `privileged`. In most cases, you can specify the exact privileges that you need by using the specific parameters instead of using `privileged`.

This parameter maps to `Privileged` in the docker create-container command and the `--privileged` option to docker run.

> **Note**
>
> This parameter is not supported for Windows containers or tasks using the Fargate launch type.

The default is `false`.

```
"privileged": true|false
```

`user`

Type: String

Required: No

The user to use inside the container. This parameter maps to `User` in the docker create-container command and the `--user` option to docker run.

> **Important**
>
> When running tasks that use the `host` network mode, don't run containers using the root user (UID 0). As a security best practice, always use a non-root user.

You can specify the `user` using the following formats. If specifying a UID or GID, you must specify it as a positive integer.

- `user`
- `user:group`
- `uid`
- `uid:gid`
- `user:gid`
- `uid:group`

> **Note**
>
> This parameter is not supported for Windows containers.

```
"user": "string"
```

`dockerSecurityOptions`

Type: String array

Valid values: "no-new-privileges" | "apparmor:PROFILE" | "label:*value*" | "credentialspec:*CredentialSpecFilePath*"

Required: No

A list of strings to provide custom configuration for multiple security systems.

For Linux tasks, this parameter can be used to reference custom labels for SELinux and AppArmor multi-level security systems.

This parameter can be used to reference a credential spec file that configures a container for Active Directory authentication. For more information, see Learn how to use gMSAs for EC2 Windows containers for Amazon ECS and Using gMSA for EC2 Linux containers on Amazon ECS.

This parameter maps to `SecurityOpt` in the docker create-container command and the `--security-opt` option to docker run.

```
"dockerSecurityOptions": ["string", ...]
```

> **Note**
>
> The Amazon ECS container agent that run on a container instance must register with the `ECS_SELINUX_CAPABLE=true` or `ECS_APPARMOR_CAPABLE=true` environment variables before containers that are placed on that instance can use these security options. For more information, see [Amazon ECS container agent configuration](#).

Resource limits

`ulimits`

Type: Object array

Required: No

A list of `ulimit` values to define for a container. This value overwrites the default resource quota setting for the operating system. This parameter maps to `Ulimits` in the docker create-container command and the `--ulimit` option to docker run.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

> **Note**
>
> This parameter is not supported for Windows containers.

```
"ulimits": [
      {
        "name":
 "core"|"cpu"|"data"|"fsize"|"locks"|"memlock"|"msgqueue"|"nice"|"nofile"|"nproc"|"rss"|"rtp
        "softLimit": integer,
        "hardLimit": integer
      }
      ...
    ]
```

name

Type: String

Valid values: `"core"` | `"cpu"` | `"data"` | `"fsize"` | `"locks"` | `"memlock"` | `"msgqueue"` | `"nice"` | `"nofile"` | `"nproc"` | `"rss"` | `"rtprio"` | `"rttime"` | `"sigpending"` | `"stack"`

Required: Yes, when `ulimits` are used

The `type` of the `ulimit`.

`hardLimit`

Type: Integer

Required: Yes, when `ulimits` are used

The hard limit for the `ulimit` type. The value can be specified in bytes, seconds, or as a count, depending on the `type` of the `ulimit`.

`softLimit`

Type: Integer

Required: Yes, when `ulimits` are used

The soft limit for the `ulimit` type. The value can be specified in bytes, seconds, or as a count, depending on the `type` of the `ulimit`.

Docker labels

`dockerLabels`

Type: String to string map

Required: No

A key/value map of labels to add to the container. This parameter maps to `Labels` in the docker create-container command and the `--label` option to docker run.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

```
"dockerLabels": {"string": "string"
```

```
        ...}
```

Other container definition parameters

The following container definition parameters can be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see Creating an Amazon ECS task definition using the console.

Topics

Linux parameters

`linuxParameters`

Type: LinuxParameters object

Required: No

Linux-specific options that are applied to the container, such as KernelCapabilities.

> **Note**
>
> This parameter isn't supported for Windows containers.

```
"linuxParameters": {
      "capabilities": {
        "add": ["string", ...],
        "drop": ["string", ...]
        }
      }
```

`capabilities`

Type: [KernelCapabilities](#) object

Required: No

The Linux capabilities for the container that are added to or dropped from the default configuration provided by Docker. For more information about these Linux capabilities, see the [capabilities(7)](#) Linux manual page.

`add`

Type: String array

Valid values: `"ALL" | "AUDIT_CONTROL" | "AUDIT_READ" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"`

Required: No

The Linux capabilities for the container to add to the default configuration provided by Docker. This parameter maps to `CapAdd` in the docker create-container command and the `--cap-add` option to docker run.

`add`

Type: String array

Valid values: `"SYS_PTRACE"`

Required: No

The Linux capabilities for the container to add to the default configuration that's provided by Docker. This parameter maps to `CapAdd` in the docker create-container command and the `--cap-add` option to docker run.

`drop`

Type: String array

Valid values: "ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"

Required: No

The Linux capabilities for the container to remove from the default configuration that's provided by Docker. This parameter maps to `CapDrop` in the docker create-container command and the `--cap-drop` option to docker run.

`devices`

Any host devices to expose to the container. This parameter maps to `Devices` in the docker create-container command and the `--device` option to docker run.

Type: Array of Device objects

Required: No

`hostPath`

The path for the device on the host container instance.

Type: String

Required: Yes

`containerPath`

The path inside the container to expose the host device at.

Type: String

Required: No

`permissions`

The explicit permissions to provide to the container for the device. By default, the container has permissions for `read`, `write`, and `mknod` on the device.

Type: Array of strings

Valid Values: `read` | `write` | `mknod`

`initProcessEnabled`

Run an `init` process inside the container that forwards signals and reaps processes. This parameter maps to the `--init` option to docker run.

This parameter requires version 1.25 of the Docker Remote API or greater on your container instance.

`maxSwap`

The total amount of swap memory (in MiB) a container can use. This parameter is translated to the `--memory-swap` option to docker run where the value is the sum of the container memory plus the `maxSwap` value.

If a `maxSwap` value of `0` is specified, the container doesn't use swap. Accepted values are `0` or any positive integer. If the `maxSwap` parameter is omitted, the container uses the swap configuration for the container instance that it's running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

`sharedMemorySize`

The value for the size (in MiB) of the `/dev/shm` volume. This parameter maps to the `--shm-size` option to docker run.

Type: Integer

`swappiness`

You can use this parameter to tune a container's memory swappiness behavior. A `swappiness` value of `0` prevents swapping from happening unless required. A `swappiness` value of `100` causes pages to be swapped frequently. Accepted values are whole numbers between `0` and `100`. If you don't specify a value, the default value of `60` is used. Moreover, if you don't specify a value for `maxSwap`, then this parameter is ignored. This parameter maps to the `--memory-swappiness` option to docker run.

> **Note**
>
> If you're using tasks on Amazon Linux 2023 the `swappiness` parameter isn't supported.

`tmpfs`

The container path, mount options, and maximum size (in MiB) of the tmpfs mount. This parameter maps to the `--tmpfs` option to docker run.

Type: Array of Tmpfs objects

Required: No

`containerPath`

The absolute file path where the tmpfs volume is to be mounted.

Type: String

Required: Yes

`mountOptions`

The list of tmpfs volume mount options.

Type: Array of strings

Required: No

Valid Values: `"defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"`

`size`

The maximum size (in MiB) of the tmpfs volume.

Type: Integer

Required: Yes

Container dependency

`dependsOn`

Type: Array of ContainerDependency objects

Required: No

The dependencies defined for container startup and shutdown. A container can contain multiple dependencies. When a dependency is defined for container startup, for container shutdown it is reversed. For an example, see Container dependency.

> **Note**
>
> If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

The instances require at least version `1.26.0` of the container agent to enable container dependencies. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS container agent. If you're using an Amazon ECS-optimized Amazon Linux AMI, your instance needs at least version `1.26.0-1` of the `ecs-init` package. If your container instances are launched from version `20190301` or later, they contain the required versions of the container agent and `ecs-init`. For more information, see Amazon ECS-optimized Linux AMIs.

```
"dependsOn": [
    {
        "containerName": "string",
        "condition": "string"
    }
]
```

`containerName`

Type: String

Required: Yes

The container name that must meet the specified condition.

`condition`

Type: String

Required: Yes

The dependency condition of the container. The following are the available conditions and their behavior:

- `START` – This condition emulates the behavior of links and volumes today. The condition validates that a dependent container is started before permitting other containers to start.
- `COMPLETE` – This condition validates that a dependent container runs to completion (exits) before permitting other containers to start. This can be useful for non-essential containers that run a script and then exit. This condition can't be set on an essential container.
- `SUCCESS` – This condition is the same as `COMPLETE`, but it also requires that the container exits with a `zero` status. This condition can't be set on an essential container.
- `HEALTHY` – This condition validates that the dependent container passes its container health check before permitting other containers to start. This requires that the dependent container has health checks configured in the task definition. This condition is confirmed only at task startup.

Container timeouts

`startTimeout`

Type: Integer

Required: No

Example values: `120`

Time duration (in seconds) to wait before giving up on resolving dependencies for a container.

For example, you specify two containers in a task definition with `containerA` having a dependency on `containerB` reaching a `COMPLETE`, `SUCCESS`, or `HEALTHY` status. If a `startTimeout` value is specified for `containerB` and it doesn't reach the desired status within that time, then `containerA` doesn't start.

> **Note**
>
> If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

The maximum value is 120 seconds.

`stopTimeout`

Type: Integer

Required: No

Example values: `120`

Time duration (in seconds) to wait before the container is forcefully killed if it doesn't exit normally on its own.

If the `stopTimeout` parameter isn't specified, the value set for the Amazon ECS container agent configuration variable `ECS_CONTAINER_STOP_TIMEOUT` is used. If neither the `stopTimeout` parameter or the `ECS_CONTAINER_STOP_TIMEOUT` agent configuration variable is set, the default values of 30 seconds for Linux containers and 30 seconds on Windows containers are used. Container instances require at least version 1.26.0 of the container agent to enable a container stop timeout value. However, we recommend using the latest container agent version. For information about how to check your agent version and update to the latest version, see Updating the Amazon ECS container agent. If you're using an Amazon ECS-optimized Amazon Linux AMI, your instance needs at least version 1.26.0-1 of the `ecs-init` package. If your container instances are launched from version `20190301` or later, they contain the required versions of the container agent and `ecs-init`. For more information, see Amazon ECS-optimized Linux AMIs.

System controls

`systemControls`

Type: SystemControl object

Required: No

A list of namespace kernel parameters to set in the container. This parameter maps to `Sysctls` in the docker create-container commandand the `--sysctl` option to docker run. For example, you can configure `net.ipv4.tcp_keepalive_time` setting to maintain longer lived connections.

We don't recommend that you specify network-related `systemControls` parameters for multiple containers in a single task that also uses either the `awsvpc` or `host` network mode. Doing this has the following disadvantages:

- For tasks that use the `awsvpc` network mode, if you set `systemControls` for any container, it applies to all containers in the task. If you set different `systemControls` for multiple containers in a single task, the container that's started last determines which `systemControls` take effect.
- For tasks that use the `host` network mode, the network namespace `systemControls` aren't supported.

If you're setting an IPC resource namespace to use for the containers in the task, the following conditions apply to your system controls. For more information, see IPC mode.

- For tasks that use the `host` IPC mode, IPC namespace `systemControls` aren't supported.
- For tasks that use the `task` IPC mode, IPC namespace `systemControls` values apply to all containers within a task.

> **Note**
>
> This parameter is not supported for Windows containers.

```
"systemControls": [
    {
         "namespace":"string",
         "value":"string"
    }
]
```

`namespace`

Type: String

Required: No

The namespace kernel parameter to set a `value` for.

Valid IPC namespace values: `"kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmax" | "kernel.shmmni" | "kernel.shm_rmid_forced"`, and `Sysctls` that start with `"fs.mqueue.*"`

Valid network namespace values: `Sysctls` that start with `"net.*"`

`value`

Type: String

Required: No

The value for the namespace kernel parameter that's specified in `namespace`.

Interactive

`interactive`

Type: Boolean

Required: No

When this parameter is `true`, you can deploy containerized applications that require `stdin` or a `tty` to be allocated. This parameter maps to `OpenStdin` in the docker create-container command and the `--interactive` option to docker run.

The default is `false`.

Pseudo terminal

`pseudoTerminal`

Type: Boolean

Required: No

When this parameter is `true`, a TTY is allocated. This parameter maps to `Tty` in the docker create-container command and the `--tty` option to docker run.

The default is `false`.

Elastic Inference accelerator name

The Elastic Inference accelerator resource requirement for your task definition.

> **Note**
>
> Amazon Elastic Inference (EI) is no longer available to customers.

The following parameters are allowed in a task definition:

`deviceName`

Type: String

Required: Yes

The Elastic Inference accelerator device name. The `deviceName` must also be referenced in a container definition see [`Elastic Inference accelerator`](#).

`deviceType`

Type: String

Required: Yes

The Elastic Inference accelerator to use.

Task placement constraints

When you register a task definition, you can provide task placement constraints that customize how Amazon ECS places tasks.

You can use constraints to place tasks based on Availability Zone, instance type, or custom attributes. For more information, see [Define which container instances Amazon ECS uses for tasks](#).

The following parameters are allowed in a container definition:

`expression`

Type: String

Required: No

A cluster query language expression to apply to the constraint. For more information, see [Create expressions to define container instances for Amazon ECS tasks](#).

`type`

Type: String

Required: Yes

The type of constraint. Use `memberOf` to restrict the selection to a group of valid candidates.

Proxy configuration

`proxyConfiguration`

Type: [ProxyConfiguration](#) object

Required: No

The configuration details for the App Mesh proxy.

For tasks that use the EC2 launch type, the container instances require at least version 1.26.0 of the container agent and at least version 1.26.0-1 of the `ecs-init` package to enable a proxy configuration. If your container instances are launched from the Amazon ECS-optimized AMI version `20190301` or later, then they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized Linux AMIs](#).

> **Note**
>
> This parameter is not supported for Windows containers.

```
"proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "string",
    "properties": [
```

```
        {
            "name": "string",
            "value": "string"
        }
    ]
}
```

`type`

Type: String

Value values: `APPMESH`

Required: No

The proxy type. The only supported value is `APPMESH`.

`containerName`

Type: String

Required: Yes

The name of the container that serves as the App Mesh proxy.

`properties`

Type: Array of [KeyValuePair](#) objects

Required: No

The set of network configuration parameters to provide the Container Network Interface (CNI) plugin, specified as key-value pairs.

- `IgnoredUID` – (Required) The user ID (UID) of the proxy container as defined by the `user` parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredGID` is specified, this field can be empty.
- `IgnoredGID` – (Required) The group ID (GID) of the proxy container as defined by the `user` parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredUID` is specified, this field can be empty.
- `AppPorts` – (Required) The list of ports that the application uses. Network traffic to these ports is forwarded to the `ProxyIngressPort` and `ProxyEgressPort`.
- `ProxyIngressPort` – (Required) Specifies the port that incoming traffic to the `AppPorts` is directed to.

- `ProxyEgressPort` – (Required) Specifies the port that outgoing traffic from the `AppPorts` is directed to.
- `EgressIgnoredPorts` – (Required) The outbound traffic going to these specified ports is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.
- `EgressIgnoredIPs` – (Required) The outbound traffic going to these specified IP addresses is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.

`name`

Type: String

Required: No

The name of the key-value pair.

`value`

Type: String

Required: No

The value of the key-value pair.

Volumes

When you register a task definition, you can optionally specify a list of volumes to be passed to the Docker daemon on a container instance, which then becomes available for access by other containers on the same container instance.

The following are the types of data volumes that can be used:

- Amazon EBS volumes — Provides cost-effective, durable, high-performance block storage for data intensive containerized workloads. You can attach 1 Amazon EBS volume per Amazon ECS task when running a standalone task, or when creating or updating a service. Amazon EBS volumes are supported for Linux tasks. For more information, see Use Amazon EBS volumes with Amazon ECS.
- Amazon EFS volumes — Provides simple, scalable, and persistent file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic. It grows and shrinks automatically as you add and remove files. Your applications can have the storage that they need and when they need it. Amazon EFS volumes are supported. For more information, see Use Amazon EFS volumes with Amazon ECS.

- FSx for Windows File Server volumes — Provides fully managed Microsoft Windows file servers. These file servers are backed by a Windows file system. When using FSx for Windows File Server together with Amazon ECS, you can provision your Windows tasks with persistent, distributed, shared, and static file storage. For more information, see Use FSx for Windows File Server volumes with Amazon ECS.

  Windows containers on Fargate do not support this option.
- Docker volumes – A Docker-managed volume that is created under `/var/lib/docker/volumes` on the host Amazon EC2 instance. Docker volume drivers (also referred to as plugins) are used to integrate the volumes with external storage systems, such as Amazon EBS. The built-in `local` volume driver or a third-party volume driver can be used. Docker volumes are supported only when running tasks on Amazon EC2 instances. Windows containers support only the use of the `local` driver. To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition.
- Bind mounts – A file or directory on the host machine that is mounted into a container. Bind mount host volumes are supported. To use bind mount host volumes, specify a `host` and optional `sourcePath` value in your task definition.

For more information, see Storage options for Amazon ECS tasks.

The following parameters are allowed in a container definition.

`name`

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens (-), and underscores (_) are allowed. This name is referenced in the `sourceVolume` parameter of the container definition `mountPoints` object.

`host`

Required: No

The `host` parameter is used to tie the lifecycle of the bind mount to the host Amazon EC2 instance, rather than the task, and where it is stored. If the `host` parameter is empty, then the Docker daemon assigns a host path for your data volume, but the data is not guaranteed to persist after the containers associated with it stop running.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`.

> **Note**
>
> The `sourcePath` parameter is supported only when using tasks that are hosted on Amazon EC2 instances.

`sourcePath`

Type: String

Required: No

When the `host` parameter is used, specify a `sourcePath` to declare the path on the host Amazon EC2 instance that is presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you. If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host Amazon EC2 instance until you delete it manually. If the `sourcePath` value does not exist on the host Amazon EC2 instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

`configuredAtLaunch`

Type: Boolean

Required: No

Specifies whether a volume is configurable at launch. When set to `true`, you can configure the volume when running a standalone task, or when creating or updating a service. When set to `true`, you won't be able to provide another volume configuration in the task definition. This parameter must be set to `true` to configure an Amazon EBS volume for attachment to a task. Setting `configuredAtLaunch` to `true` and deferring volume configuration to the launch phase allows you to create task definitions that aren't constrained to a volume type or to specific volume settings. Doing this makes your task definition reusable across different execution environments. For more information, see Amazon EBS volumes.

`dockerVolumeConfiguration`

Type: DockerVolumeConfiguration Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are supported only when running tasks on EC2 instances. Windows containers support only the use of the `local` driver. To use bind mounts, specify a `host` instead.

`scope`

Type: String

Valid Values: `task` | `shared`

Required: No

The scope for the Docker volume, which determines its lifecycle. Docker volumes that are scoped to a `task` are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as `shared` persist after the task stops.

`autoprovision`

Type: Boolean

Default value: `false`

Required: No

If this value is `true`, the Docker volume is created if it doesn't already exist. This field is used only if the `scope` is `shared`. If the `scope` is `task`, then this parameter must be omitted.

`driver`

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because this name is used for task placement. If the driver was installed by using the Docker plugin CLI, use `docker plugin ls` to retrieve the driver name from your container instance. If the driver was installed by using another method, use Docker plugin discovery to retrieve the driver name.

`driverOpts`

Type: String

Required: No

A map of Docker driver-specific options to pass through. This parameter maps to `DriverOpts` in the Create a volume section of Docker.

`labels`

Type: String

Required: No

Custom metadata to add to your Docker volume.

`efsVolumeConfiguration`

Type: [EFSVolumeConfiguration](#) Object

Required: No

This parameter is specified when using Amazon EFS volumes.

`fileSystemId`

Type: String

Required: Yes

The Amazon EFS file system ID to use.

`rootDirectory`

Type: String

Required: No

The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume will be used. Specifying / has the same effect as omitting this parameter.

> **Important**
>
> If an EFS access point is specified in the `authorizationConfig`, the root directory parameter must either be omitted or set to /, which will enforce the path set on the EFS access point.

`transitEncryption`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. If Amazon EFS IAM authorization is used, transit encryption must be enabled. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see Encrypting Data in Transit in the *Amazon Elastic File System User Guide*.

`transitEncryptionPort`

Type: Integer

Required: No

The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you don't specify a transit encryption port, the task will use the port selection strategy that the Amazon EFS mount helper uses. For more information, see EFS Mount Helper in the *Amazon Elastic File System User Guide*.

`authorizationConfig`

Type: EFSAuthorizationConfig Object

Required: No

The authorization configuration details for the Amazon EFS file system.

`accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value in the `efsVolumeConfiguration` must either be omitted or set to `/`, which will enforce the path set on the EFS access point. If an access point is used, transit encryption must be enabled in the `EFSVolumeConfiguration`. For more information, see Working with Amazon EFS Access Points in the *Amazon Elastic File System User Guide*.

`iam`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to use the Amazon ECS task IAM role that's defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the `EFSVolumeConfiguration`. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see IAM Roles for Tasks.

`FSxWindowsFileServerVolumeConfiguration`

Type: FSxWindowsFileServerVolumeConfiguration Object

Required: Yes

This parameter is specified when you're using an Amazon FSx for Windows File Server file system for task storage.

`fileSystemId`

Type: String

Required: Yes

The FSx for Windows File Server file system ID to use.

`rootDirectory`

Type: String

Required: Yes

The directory within the FSx for Windows File Server file system to mount as the root directory inside the host.

`authorizationConfig`

`credentialsParameter`

Type: String

Required: Yes

The authorization credential options.

options:

- Amazon Resource Name (ARN) of an AWS Secrets Manager secret.
- ARN of an AWS Systems Manager parameter.

`domain`

Type: String

Required: Yes

A fully qualified domain name hosted by an AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) directory or a self-hosted EC2 Active Directory.

Tags

When you register a task definition, you can optionally specify metadata tags that are applied to the task definition. Tags help you categorize and organize your task definition. Each tag consists of a key and an optional value. You define both of them. For more information, see Tagging Amazon ECS resources.

> **Important**
>
> Don't add personally identifiable information or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags aren't intended to be used for private or sensitive data.

The following parameters are allowed in a tag object.

`key`

Type: String

Required: No

One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

`value`

Type: String

Required: No

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

Other task definition parameters

The following task definition parameters can be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see Creating an Amazon ECS task definition using the console.

Topics

- IPC mode
- PID mode
- Fault injection

IPC mode

`ipcMode`

Type: String

Required: No

The IPC resource namespace to use for the containers in the task. The valid values are `host`, `task`, or `none`. If `host` is specified, then all the containers that are within the tasks that specified the `host` IPC mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If `task` is specified, all the containers that are within the specified task share the same IPC resources. If `none` is specified, then IPC resources within the containers of a task are private and not shared with other containers in a task or on the container instance. If no value is specified, then the IPC resource namespace sharing depends on the Docker daemon setting on the container instance.

If the `host` IPC mode is used, there's a heightened risk of undesired IPC namespace exposure.

If you're setting namespaced kernel parameters that use `systemControls` for the containers in the task, the following applies to your IPC resource namespace.

- For tasks that use the `host` IPC mode, IPC namespace that's related `systemControls` aren't supported.
- For tasks that use the `task` IPC mode, `systemControls` that relate to the IPC namespace apply to all containers within a task.

PID mode

`pidMode`

Type: String

Valid Values: `host` | `task`

Required: No

The process namespace to use for the containers in the task. The valid values are `host` or `task`. For example, monitoring sidecars might need `pidMode` to access information about other containers running in the same task.

If `host` is specified, all containers within the tasks that specified the `host` PID mode on the same container instance share the same process namespace with the host Amazon EC2 instance.

If `task` is specified, all containers within the specified task share the same process namespace.

If no value is specified, the default is a private namespace for each container.

If the `host` PID mode is used, there's a heightened risk of undesired process namespace exposure.

> **Note**
>
> This parameter is not supported for Windows containers.

Fault injection

`enableFaultInjection`

Type: Boolean

Valid Values: `true` | `false`

Required: No

If this parameter is set to `true`, in a task's payload, Amazon ECS accepts fault injection requests from the task's containers. By default, this parameter is set to `false`.

Amazon ECS task definition template

An empty task definition template is shown as follows. You can use this template to create your task definition, which can then be pasted into the console JSON input area or saved to a file and used with the AWS CLI `--cli-input-json` option. For more information, see Amazon ECS task definition parameters for the Fargate launch type.

Amazon EC2 launch type template

```
{
  "family": "",
  "taskRoleArn": "",
  "executionRoleArn": "",
  "networkMode": "none",
  "containerDefinitions": [
    {
      "name": "",
      "image": "",
      "repositoryCredentials": {
        "credentialsParameter": ""
      },
      "cpu": 0,
      "memory": 0,
      "memoryReservation": 0,
      "links": [""],
      "portMappings": [
        {
          "containerPort": 0,
          "hostPort": 0,
          "protocol": "tcp"
        }
      ],
      "restartPolicy": {
        "enabled": true,
        "ignoredExitCodes": [0],
        "restartAttemptPeriod": 180
      },
```

```
      "essential": true,
      "entryPoint": [""],
      "command": [""],
      "environment": [
        {
          "name": "",
          "value": ""
        }
      ],
      "environmentFiles": [
        {
          "value": "",
          "type": "s3"
        }
      ],
      "mountPoints": [
        {
          "sourceVolume": "",
          "containerPath": "",
          "readOnly": true
        }
      ],
      "volumesFrom": [
        {
          "sourceContainer": "",
          "readOnly": true
        }
      ],
      "linuxParameters": {
        "capabilities": {
          "add": [""],
          "drop": [""]
        },
        "devices": [
          {
            "hostPath": "",
            "containerPath": "",
            "permissions": ["read"]
          }
        ],
        "initProcessEnabled": true,
        "sharedMemorySize": 0,
        "tmpfs": [
          {
```

```
            "containerPath": "",
            "size": 0,
            "mountOptions": [""]
          }
        ],
        "maxSwap": 0,
        "swappiness": 0
      },
      "secrets": [
        {
          "name": "",
          "valueFrom": ""
        }
      ],
      "dependsOn": [
        {
          "containerName": "",
          "condition": "COMPLETE"
        }
      ],
      "startTimeout": 0,
      "stopTimeout": 0,
      "hostname": "",
      "user": "",
      "workingDirectory": "",
      "disableNetworking": true,
      "privileged": true,
      "readonlyRootFilesystem": true,
      "dnsServers": [""],
      "dnsSearchDomains": [""],
      "extraHosts": [
        {
          "hostname": "",
          "ipAddress": ""
        }
      ],
      "dockerSecurityOptions": [""],
      "interactive": true,
      "pseudoTerminal": true,
      "dockerLabels": {
        "KeyName": ""
      },
      "ulimits": [
        {
```

```
          "name": "nofile",
          "softLimit": 0,
          "hardLimit": 0
        }
      ],
      "logConfiguration": {
        "logDriver": "splunk",
        "options": {
          "KeyName": ""
        },
        "secretOptions": [
          {
            "name": "",
            "valueFrom": ""
          }
        ]
      },
      "healthCheck": {
        "command": [""],
        "interval": 0,
        "timeout": 0,
        "retries": 0,
        "startPeriod": 0
      },
      "systemControls": [
        {
          "namespace": "",
          "value": ""
        }
      ],
      "resourceRequirements": [
        {
          "value": "",
          "type": "InferenceAccelerator"
        }
      ],
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "KeyName": ""
        }
      }
    }
  ],
```

```
  "volumes": [
    {
      "name": "",
      "host": {
        "sourcePath": ""
      },
      "configuredAtLaunch": true,
      "dockerVolumeConfiguration": {
        "scope": "shared",
        "autoprovision": true,
        "driver": "",
        "driverOpts": {
          "KeyName": ""
        },
        "labels": {
          "KeyName": ""
        }
      },
      "efsVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "transitEncryption": "DISABLED",
        "transitEncryptionPort": 0,
        "authorizationConfig": {
          "accessPointId": "",
          "iam": "ENABLED"
        }
      },
      "fsxWindowsFileServerVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "authorizationConfig": {
          "credentialsParameter": "",
          "domain": ""
        }
      }
    }
  ],
  "placementConstraints": [
    {
      "type": "memberOf",
      "expression": ""
    }
  ],
```

```
  "requiresCompatibilities": ["EC2"],
  "cpu": "",
  "memory": "",
  "tags": [
    {
      "key": "",
      "value": ""
    }
  ],
  "pidMode": "task",
  "ipcMode": "task",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "",
    "properties": [
      {
        "name": "",
        "value": ""
      }
    ]
  },
  "inferenceAccelerators": [
    {
      "deviceName": "",
      "deviceType": ""
    }
  ],
  "ephemeralStorage": {
    "sizeInGiB": 0
  },
  "runtimePlatform": {
    "cpuArchitecture": "X86_64",
    "operatingSystemFamily": "WINDOWS_SERVER_20H2_CORE"
  }
}
```

Fargate launch type template

Important

For the Fargate launch type, you must include the `operatingSystemFamily` parameter with one of the following values:

- LINUX

- WINDOWS_SERVER_2019_FULL
- WINDOWS_SERVER_2019_CORE
- WINDOWS_SERVER_2022_FULL
- WINDOWS_SERVER_2022_CORE

```
{
    "family": "",
    "runtimePlatform": {"operatingSystemFamily": ""},
    "taskRoleArn": "",
    "executionRoleArn": "",
    "networkMode": "awsvpc",
    "platformFamily": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "repositoryCredentials": {"credentialsParameter": ""},
            "cpu": 0,
            "memory": 0,
            "memoryReservation": 0,
            "links": [""],
            "portMappings": [
                {
                    "containerPort": 0,
                    "hostPort": 0,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [""],
            "command": [""],
            "environment": [
                {
                    "name": "",
                    "value": ""
                }
            ],
```

```
            "environmentFiles": [
                {
                    "value": "",
                    "type": "s3"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
                }
            ],
            "volumesFrom": [
                {
                    "sourceContainer": "",
                    "readOnly": true
                }
            ],
            "linuxParameters": {
                "capabilities": {
                    "add": [""],
                    "drop": [""]
                },
                "devices": [
                    {
                        "hostPath": "",
                        "containerPath": "",
                        "permissions": ["read"]
                    }
                ],
                "initProcessEnabled": true,
                "sharedMemorySize": 0,
                "tmpfs": [
                    {
                        "containerPath": "",
                        "size": 0,
                        "mountOptions": [""]
                    }
                ],
                "maxSwap": 0,
                "swappiness": 0
            },
            "secrets": [
```

```
                {
                    "name": "",
                    "valueFrom": ""
                }
            ],
            "dependsOn": [
                {
                    "containerName": "",
                    "condition": "HEALTHY"
                }
            ],
            "startTimeout": 0,
            "stopTimeout": 0,
            "hostname": "",
            "user": "",
            "workingDirectory": "",
            "disableNetworking": true,
            "privileged": true,
            "readonlyRootFilesystem": true,
            "dnsServers": [""],
            "dnsSearchDomains": [""],
            "extraHosts": [
                {
                    "hostname": "",
                    "ipAddress": ""
                }
            ],
            "dockerSecurityOptions": [""],
            "interactive": true,
            "pseudoTerminal": true,
            "dockerLabels": {"KeyName": ""},
            "ulimits": [
                {
                    "name": "msgqueue",
                    "softLimit": 0,
                    "hardLimit": 0
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {"KeyName": ""},
                "secretOptions": [
                    {
                        "name": "",
```

```
                        "valueFrom": ""
                    }
                ]
            },
            "healthCheck": {
                "command": [""],
                "interval": 0,
                "timeout": 0,
                "retries": 0,
                "startPeriod": 0
            },
            "systemControls": [
                {
                    "namespace": "",
                    "value": ""
                }
            ],
            "resourceRequirements": [
                {
                    "value": "",
                    "type": "GPU"
                }
            ],
            "firelensConfiguration": {
                "type": "fluentd",
                "options": {"KeyName": ""}
            }
        }
    ],
    "volumes": [
        {
            "name": "",
            "host": {"sourcePath": ""},
            "configuredAtLaunch":true,
            "dockerVolumeConfiguration": {
                "scope": "task",
                "autoprovision": true,
                "driver": "",
                "driverOpts": {"KeyName": ""},
                "labels": {"KeyName": ""}
            },
            "efsVolumeConfiguration": {
                "fileSystemId": "",
                "rootDirectory": "",
```

```
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": 0,
                "authorizationConfig": {
                    "accessPointId": "",
                    "iam": "ENABLED"
                }
            }
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "cpu": "",
    "memory": "",
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "ephemeralStorage": {"sizeInGiB": 0},
    "pidMode": "task",
    "ipcMode": "none",
    "proxyConfiguration": {
        "type": "APPMESH",
        "containerName": "",
        "properties": [
            {
                "name": "",
                "value": ""
            }
        ]
    },
    "inferenceAccelerators": [
        {
            "deviceName": "",
            "deviceType": ""
        }
    ]
}
```

You can generate this task definition template using the following AWS CLI command.

```
aws ecs register-task-definition --generate-cli-skeleton
```

Example Amazon ECS task definitions

You can copy the examples and snippets to start creating your own task definitions.

You can copy the examples, and then paste them when you use the **Configure via JSON** option in the console. Make sure to customize the examples, such as using your account ID. You can include the snippets in your task definition JSON. For more information, see Creating an Amazon ECS task definition using the console and Amazon ECS task definition parameters for the Fargate launch type.

For more task definition examples, see AWS Sample Task Definitions on GitHub.

Topics

Webserver

The following is an example task definition using the Linux containers on Fargate launch type that sets up a web server:

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
```

```
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
         "essential": true,
         "image": "public.ecr.aws/docker/library/httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
         "name": "sample-fargate-app",
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
    },
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

The following is an example task definition using the Windows containers on Fargate launch type that sets up a web server:

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

splunk log driver

The following snippet demonstrates how to use the `splunk` log driver in a task definition that sends the logs to a remote service. The Splunk token parameter is specified as a secret option because it can be treated as sensitive data. For more information, see Pass sensitive data to an Amazon ECS container.

```
"containerDefinitions": [{
  "logConfiguration": {
   "logDriver": "splunk",
   "options": {
    "splunk-url": "https://cloud.splunk.com:8080",
    "tag": "tag_name",
   },
   "secretOptions": [{
    "name": "splunk-token",
    "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:splunk-token-KnrBkD"
}],
```

fluentd log driver

The following snippet demonstrates how to use the `fluentd` log driver in a task definition that sends the logs to a remote service. The `fluentd-address` value is specified as a secret option as it may be treated as sensitive data. For more information, see Pass sensitive data to an Amazon ECS container.

```
"containerDefinitions": [{
 "logConfiguration": {
  "logDriver": "fluentd",
  "options": {
   "tag": "fluentd demo"
  },
  "secretOptions": [{
   "name": "fluentd-address",
   "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:fluentd-address-KnrBkD"
  }]
 },
 "entryPoint": [],
 "portMappings": [{
             "hostPort": 80,
             "protocol": "tcp",
             "containerPort": 80
             },
             {
```

```
  "hostPort": 24224,
  "protocol": "tcp",
  "containerPort": 24224
 }]
}],
```

gelf log driver

The following snippet demonstrates how to use the `gelf` log driver in a task definition that sends the logs to a remote host running Logstash that takes Gelf logs as an input. For more information, see `logConfiguration`.

```
"containerDefinitions": [{
 "logConfiguration": {
  "logDriver": "gelf",
  "options": {
   "gelf-address": "udp://logstash-service-address:5000",
   "tag": "gelf task demo"
  }
 },
 "entryPoint": [],
 "portMappings": [{
   "hostPort": 5000,
   "protocol": "udp",
   "containerPort": 5000
  },
  {
   "hostPort": 5000,
   "protocol": "tcp",
   "containerPort": 5000
  }
 ]
}],
```

Workloads on external instances

When registering an Amazon ECS task definition, use the `requiresCompatibilities` parameter and specify `EXTERNAL` which validates that the task definition is compatible to use when running Amazon ECS workloads on your external instances. If you use the console for registering a task definition, you must use the JSON editor. For more information, see Creating an Amazon ECS task definition using the console.

> **Important**
>
> If your tasks require a task execution IAM role, make sure that it's specified in the task definition.

When you deploy your workload, use the EXTERNAL launch type when creating your service or running your standalone task.

The following is an example task definition.

Linux

```
{
 "requiresCompatibilities": [
  "EXTERNAL"
 ],
 "containerDefinitions": [{
  "name": "nginx",
  "image": "public.ecr.aws/nginx/nginx:latest",
  "memory": 256,
  "cpu": 256,
  "essential": true,
  "portMappings": [{
   "containerPort": 80,
   "hostPort": 8080,
   "protocol": "tcp"
  }]
 }],
 "networkMode": "bridge",
 "family": "nginx"
}
```

Windows

```
{
 "requiresCompatibilities": [
  "EXTERNAL"
 ],
 "containerDefinitions": [{
  "name": "windows-container",
  "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
```

```
      "memory": 256,
      "cpu": 512,
      "essential": true,
      "portMappings": [{
       "containerPort": 80,
       "hostPort": 8080,
       "protocol": "tcp"
      }]
     }],
     "networkMode": "bridge",
     "family": "windows-container"
    }
```

Amazon ECR image and task definition IAM role

The following snippet uses an Amazon ECR image called `aws-nodejs-sample` with the `v1` tag from the `123456789012.dkr.ecr.us-west-2.amazonaws.com` registry. The container in this task inherits IAM permissions from the `arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole` role. For more information, see [Amazon ECS task IAM role](#).

```
{
    "containerDefinitions": [
        {
            "name": "sample-app",
            "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-sample:v1",
            "memory": 200,
            "cpu": 10,
            "essential": true
        }
    ],
    "family": "example_task_3",
    "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
}
```

Entrypoint with command

The following snippet demonstrates the syntax for a Docker container that uses an entry point and a command argument. This container pings `example.com` four times and then exits.

```
{
```

```
    "containerDefinitions": [
        {
            "memory": 32,
            "essential": true,
            "entryPoint": ["ping"],
            "name": "alpine_ping",
            "readonlyRootFilesystem": true,
            "image": "alpine:3.4",
            "command": [
                "-c",
                "4",
                "example.com"
            ],
            "cpu": 16
        }
    ],
    "family": "example_task_2"
}
```

Container dependency

This snippet demonstrates the syntax for a task definition with multiple containers where container dependency is specified. In the following task definition, the `envoy` container must reach a healthy status, determined by the required container health check parameters, before the `app` container will start. For more information, see Container dependency.

```
{
  "family": "appmesh-gateway",
  "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
  },
  "proxyConfiguration":{
      "type": "APPMESH",
      "containerName": "envoy",
      "properties": [
          {
              "name": "IgnoredUID",
              "value": "1337"
          },
          {
              "name": "ProxyIngressPort",
              "value": "15000"
          },
```

```
          {
              "name": "ProxyEgressPort",
              "value": "15001"
          },
          {
              "name": "AppPorts",
              "value": "9080"
          },
          {
              "name": "EgressIgnoredIPs",
              "value": "169.254.170.2,169.254.169.254"
          }
      ]
  },
  "containerDefinitions": [
    {
      "name": "app",
      "image": "application_image",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
      "name": "envoy",
      "image": "840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-
envoy:v1.15.1.0-prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/meshName/virtualNode/virtualNodeName"
        },
        {
```

```
          "name": "ENVOY_LOG_LEVEL",
          "value": "info"
        }
      ],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "echo hello"
        ],
        "interval": 5,
        "timeout": 2,
        "retries": 3
      }
    }
  ],
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

Volumes in task definitions

Use the following to understand how to specify volumes in tasks.

- For information about how to configure an Amazon EBS volume, see Specify Amazon EBS volume configuration at Amazon ECS deployment.
- For information about how to configure an Amazon EFS volume, see Configuring Amazon EFS file systems for Amazon ECS using the console.
- For information about how to configure a FSx for Windows File Server volume, see Learn how to configure FSx for Windows File Server file systems for Amazon ECS.
- For information about how to configure a Docker volume, see Docker volume examples for Amazon ECS.
- For information about how to configure a bind mount, see Bind mount examples for Amazon ECS.

Windows sample task definitions

The following is a sample task definition to help you get started with Windows containers on Amazon ECS.

Example Amazon ECS Console Sample Application for Windows

The following task definition is the Amazon ECS console sample application that is produced in the first-run wizard for Amazon ECS; it has been ported to use the `microsoft/iis` Windows container image.

```
{
  "family": "windows-simple-iis",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 1024,
      "entryPoint":["powershell", "-Command"],
      "command":["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -
Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "memory": 1024,
      "essential": true
    }
  ],
  "networkMode": "awsvpc",
  "memory": "1024",
  "cpu": "1024"
}
```

Amazon ECS clusters

An Amazon ECS cluster is a logical grouping of tasks or services. In addition to tasks and services, a cluster consists of the following resources:

- The infrastructure capacity which can be a combination of the following:
  - Amazon EC2 instances in the AWS cloud
  - Serverless (AWS Fargate) in the AWS cloud
  - On-premises virtual machines (VM) or servers
- The network (VPC and subnet) where your tasks and services run

  When you use Amazon EC2 instances for the capacity, the subnet can be in Availability Zones, Local Zones, Wavelength Zones or AWS Outposts.
- An optional namespace

  The namespace is used for service-to-service communication with Service Connect.
- A monitoring option

  CloudWatch Container Insights comes at an additional cost and is a fully managed service. It automatically collects, aggregates, and summarizes Amazon ECS metrics and logs.

The following are general concepts about Amazon ECS clusters.

- You create clusters to separate your resources.
- Clusters are AWS Region specific.
- Clusters can be in any of the following states.

  ACTIVE

    The cluster is ready to accept tasks and, if applicable, you can register container instances with the cluster.

  PROVISIONING

    The cluster has capacity providers associated with it and the resources needed for the capacity provider are being created.

DEPROVISIONING

The cluster has capacity providers associated with it and the resources needed for the capacity provider are being deleted.

FAILED

The cluster has capacity providers associated with it and the resources needed for the capacity provider have failed to create.

INACTIVE

The cluster has been deleted. Clusters with an `INACTIVE` status may remain discoverable in your account for a period of time. This behavior is subject to change in the future, so make sure you do not rely on `INACTIVE` clusters persisting.

- You can use different instance types for the EC2 launch type or Auto Scaling group capacity providers. An instance can only be registered to one cluster at a time.
- You can restrict access to clusters by creating custom IAM policies. For information, see Amazon ECS cluster examples section in Identity-based policy examples for Amazon Elastic Container Service.
- You can use Service Auto Scaling to scale Fargate tasks. For more information, see Automatically scale your Amazon ECS service.
- You can configure a default Service Connect namespace for a cluster. After you set a default Service Connect namespace, any new services created in the cluster can be added as client services in the namespace by turning on Service Connect. No additional configuration is required. For more information, see Use Service Connect to connect Amazon ECS services with short names.

Capacity providers

Amazon ECS capacity providers manage the scaling of infrastructure for tasks in your clusters. Each cluster can have one or more capacity providers and an optional capacity provider strategy. You can assign a default capacity provider strategy to the cluster. The capacity provider strategy determines how the tasks are spread across the cluster's capacity providers. When you run a standalone task or create a service, you either use the cluster's default capacity provider strategy or a capacity provider strategy that overrides the default one.The cluster's default capacity provider strategy only applies when you don't specify a launch type, or capacity provider strategy for your task or service. If you provide either of these parameters, the default strategy isn't used.

For Fargate, you do not need to create or manage the capacity. You just need to associate any of the following pre-defined capacity providers with the cluster:

- Fargate
- Fargate Spot

When you use EC2 instances for your capacity, you use Auto Scaling group to manage the EC2 instances. Auto Scaling helps ensure that you have the correct number of EC2 instances available to handle the application load.

A cluster can contain a mix of tasks that are hosted on AWS Fargate, Amazon EC2 instances, or external instances. Tasks can run on Fargate or EC2 infrastructure as a launch type or a capacity provider strategy. If you use EC2 as a launch type, Amazon ECS doesn't track and scale the capacity of Amazon EC2 Auto Scaling groups. For more information about launch types, see Amazon ECS launch types.

A cluster can contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers. A capacity provider strategy can only contain Auto Scaling group capacity providers or Fargate capacity providers.

Amazon ECS clusters for Fargate

Amazon ECS capacity providers manage the scaling of infrastructure for tasks in your clusters. Each cluster can have one or more capacity providers and an optional capacity provider strategy. The capacity provider strategy determines how the tasks are spread across the cluster's capacity providers. When you run a standalone task or create a service, you either use the cluster's default capacity provider strategy or a capacity provider strategy that overrides the default one.

When you run your tasks on AWS Fargate, you do not need to create or manage the capacity. You just need to associate any of the following pre-defined capacity providers with the cluster:

- Fargate
- Fargate Spot

With Amazon ECS on AWS Fargate capacity providers, you can use both Fargate and Fargate Spot capacity with your Amazon ECS tasks.

With Fargate Spot, you can run interruption tolerant Amazon ECS tasks at a rate that's discounted compared to the Fargate price. Fargate Spot runs tasks on spare compute capacity. When AWS needs the capacity back, your tasks are interrupted with a two-minute warning.

When tasks that use the Fargate and Fargate Spot capacity providers are stopped, the task state change event is sent to Amazon EventBridge. The stopped reason describes the cause. For more information, see [Amazon ECS task state change events](#).

A cluster can contain a mix of Fargate and Auto Scaling group capacity providers. However, a capacity provider strategy can only contain either Fargate or Auto Scaling group capacity providers, but not both. For more information, see [Auto Scaling Group Capacity Providers](#).

Consider the following when using capacity providers:

- You must associate a capacity provider with a cluster before you associate it with the capacity provider strategy.
- You can specify a maximum of 20 capacity providers for a capacity provider strategy.
- You can't update a service using an Auto Scaling group capacity provider to use a Fargate capacity provider. The opposite is also the case.
- In a capacity provider strategy, if no `weight` value is specified for a capacity provider in the console, then the default value of `1` is used. If using the API or AWS CLI, the default value of `0` is used.
- When multiple capacity providers are specified within a capacity provider strategy, at least one of the capacity providers must have a weight value that's greater than zero. Any capacity providers with a weight of zero aren't used to place tasks. If you specify multiple capacity providers in a strategy with all the same weight of zero, then any `RunTask` or `CreateService` actions using the capacity provider strategy fail.
- In a capacity provider strategy, only one capacity provider can have a defined *base* value. If no base value is specified, the default value of zero is used.
- A cluster can contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers. However, a capacity provider strategy can only contain Auto Scaling group or Fargate capacity providers, but not both.
- A cluster can contain a mix of services and standalone tasks that use both capacity providers and launch types. A service can be updated to use a capacity provider strategy rather than a launch type. However, you must force a new deployment when doing so.

Fargate Spot termination notices

During periods of extremely high demand, Fargate Spot capacity might be unavailable. This can cause Fargate Spot tasks to be delayed. When this happens, Amazon ECS services retry launching tasks until the required capacity becomes available. Fargate doesn't replace Spot capacity with on-demand capacity.

When tasks using Fargate Spot capacity are stopped due to a Spot interruption, a two-minute warning is sent before a task is stopped. The warning is sent as a task state change event to Amazon EventBridge and as a SIGTERM signal to the running task. If you use Fargate Spot as part of a service, then in this scenario the service scheduler receives the interruption signal and attempts to launch additional tasks on Fargate Spot if there's capacity available. A service with only one task is interrupted until capacity is available. For more information about a graceful shutdown, see Graceful shutdowns with ECS.

To ensure that your containers exit gracefully before the task stops, you can configure the following:

- A `stopTimeout` value of `120` seconds or less can be specified in the container definition that the task is using. The default `stopTimeout` value is 30 seconds. You can specify a longer `stopTimeout` value to give yourself more time between the moment that the task state change event is received and the point in time when the container is forcefully stopped. For more information, see Container timeouts.
- The SIGTERM signal must be received from within the container to perform any cleanup actions. Failure to process this signal results in the task receiving a SIGKILL signal after the configured `stopTimeout` and may result in data loss or corruption.

The following is a snippet of a task state change event. This snippet displays the stopped reason and stop code for a Fargate Spot interruption.

```
{
  "version": "0",
  "id": "9bcdac79-b31f-4d3d-9410-fbd727c29fab",
  "detail-type": "ECS Task State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:task/b99d40b3-5176-4f71-9a52-9dbd6f1cebef"
  ],
```

```
  "detail": {
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "createdAt": "2016-12-06T16:41:05.702Z",
    "desiredStatus": "STOPPED",
    "lastStatus": "RUNNING",
    "stoppedReason": "Your Spot Task was interrupted.",
    "stopCode": "SpotInterruption",
    "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/
b99d40b3-5176-4f71-9a52-9dbd6fEXAMPLE",
    ...
  }
}
```

The following is an event pattern that's used to create an EventBridge rule for Amazon ECS task state change events. You can optionally specify a cluster in the `detail` field. Doing so means that you will receive task state change events for that cluster. For more information about creating an EventBridge rule, see Getting started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Task State Change"
    ],
    "detail": {
        "clusterArn": [
            "arn:aws:ecs:us-west-2:111122223333:cluster/default"
        ]
    }
}
```

Creating an Amazon ECS cluster for the Fargate launch type

You create a cluster to define the infrastructure your tasks and services run on.

Before you begin, be sure that you've completed the steps in Set up to use Amazon ECS and assign the appropriate IAM permission. For more information, see the section called “Amazon ECS cluster examples”. The Amazon ECS console creates the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack.

The console automatically associates the Fargate and Fargate Spot capacity providers with the cluster.

You can modify the following options:

- Add a namespace to the cluster.

  A namespace allows services that you create in the cluster can connect to the other services in the namespace without additional configuration. For more information, see Interconnect Amazon ECS services.
- Add tags to help you identify your cluster.
- Assign an AWS KMS key for your managed storage. For information about how to create a key, see Create a KMS key in the *AWS Key Management Service User Guide*.
- Assign an AWS KMS key for your Fargate ephemeral storage. For information about how to create a key, see Create a KMS key in the *AWS Key Management Service User Guide*.
- Configure the AWS KMS key and logging for ECS Exec.

Procedure

To create a new cluster (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, configure the following:

   - For **Cluster name**, enter a unique name.

     The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.
   - (Optional) To have the namespace used for Service Connect be different from the cluster name, for **Namespace**, enter a unique name.
6. (Optional) Use Container Insights, expand **Monitoring**, and then choose one of the following options:

   - To use the recommended Container Insights with enhanced observability, choose **Container Insights with enhanced observability**.

- To use Container Insights, choose **Container Insights**.

7. (Optional) To use ECS Exec to debug tasks in the cluster, expand **Troubleshooting configuration**, and then configure the following:
   - (Optional) For **AWS KMS key for ECS Exec**, enter the ARN of the AWS KMS key you want to use to encrypt the ECS Exec session data.
   - (Optional) For **ECS Exec logging**, choose the log destination:
     - To send logs to CloudWatch Logs, choose **Amazon CloudWatch**.
     - To send logs to Amazon S3, choose **Amazon S3**.
     - To disable logging, choose **None**.
8. (Optional), Under **Encryption**, you can configure the following:
   - Encrypt your data on Fargate ephemeral storage. Under **Encryption**, for **Fargate ephemeral storage**, enter the ARN of the AWS KMS key you want to use to encrypt the Fargate ephemeral storage data.
   - Encrypt the data on managed storage. Under **Encryption**, for **Managed storage**, enter the ARN of the AWS KMS key you want to use to encrypt the managed storage data.
9. (Optional) To help identify your cluster, expand **Tags**, and then configure your tags.

   [Add a tag] Choose **Add tag** and do the following:
   - For **Key**, enter the key name.
   - For **Value**, enter the key value.

   [Remove a tag] Choose **Remove** to the right of the tag's Key and Value.
10. Choose **Create**.

Next steps

After you create the cluster, you can create task definitions for your applications and then run them as standalone tasks, or as part of a service. For more information, see the following:

- Amazon ECS task definitions
- Running an application as an Amazon ECS task
- Creating an Amazon ECS rolling update deployment

Amazon ECS capacity providers for the EC2 launch type

When you use Amazon EC2 instances for your capacity, you use Auto Scaling groups to manage the Amazon EC2 instances registered to their clusters. Auto Scaling helps ensure that you have the correct number of Amazon EC2 instances available to handle the application load.

You can use the managed scaling feature to have Amazon ECS manage the scale-in and scale-out actions of the Auto Scaling group, or you can manage the scaling actions yourself. For more information, see Automatically manage Amazon ECS capacity with cluster auto scaling.

We recommend that you create a new empty Auto Scaling group. If you use an existing Auto Scaling group, any Amazon EC2 instances that are associated with the group that were already running and registered to an Amazon ECS cluster before the Auto Scaling group being used to create a capacity provider might not be properly registered with the capacity provider. This might cause issues when using the capacity provider in a capacity provider strategy. Use `DescribeContainerInstances` to confirm whether a container instance is associated with a capacity provider or not.

> **Note**
>
> To create an empty Auto Scaling group, set the desired count to zero. After you created the capacity provider and associated it with a cluster, you can then scale it out.
> When you use the Amazon ECS console, Amazon ECS creates an Amazon EC2 launch template and Auto Scaling group on your behalf as part of the AWS CloudFormation stack. They are prefixed with `EC2ContainerService-<*ClusterName*>`. You can use the Auto Scaling group as a capacity provider for that cluster.

We recommend you use managed instance draining to allow for graceful termination of Amazon EC2 instances that won't disrupt your workloads. This feature is on by default. For more information, see Safely stop Amazon ECS workloads running on EC2 instances

Consider the following when using Auto Scaling group capacity providers in the console:

- An Auto Scaling group must have a `MaxSize` greater than zero to scale out.
- The Auto Scaling group can't have instance weighting settings.
- If the Auto Scaling group can't scale out to accommodate the number of tasks run, the tasks fails to transition beyond the `PROVISIONING` state.

- Don't modify the scaling policy resource associated with your Auto Scaling groups that are managed by capacity providers.
- If managed scaling is turned on when you create a capacity provider, the Auto Scaling group desired count can be set to `0`. When managed scaling is turned on, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group.
- You must associate capacity provider with a cluster before you associate it with the capacity provider strategy.
- You can specify a maximum of 20 capacity providers for a capacity provider strategy.
- You can't update a service using an Auto Scaling group capacity provider to use a Fargate capacity provider. The opposite is also the case.
- In a capacity provider strategy, if no `weight` value is specified for a capacity provider in the console, then the default value of `1` is used. If using the API or AWS CLI, the default value of `0` is used.
- When multiple capacity providers are specified within a capacity provider strategy, at least one of the capacity providers must have a weight value that's greater than zero. Any capacity providers with a weight of zero aren't used to place tasks. If you specify multiple capacity providers in a strategy with all the same weight of zero, then any `RunTask` or `CreateService` actions using the capacity provider strategy fail.
- In a capacity provider strategy, only one capacity provider can have a defined *base* value. If no base value is specified, the default value of zero is used.
- A cluster can contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers. However, a capacity provider strategy can only contain Auto Scaling group or Fargate capacity providers, but not both.
- A cluster can contain a mix of services and standalone tasks that use both capacity providers and launch types. A service can be updated to use a capacity provider strategy rather than a launch type. However, you must force a new deployment when doing so.
- Amazon ECS supports Amazon EC2 Auto Scaling warm pools. A warm pool is a group of pre-initialized Amazon EC2 instances ready to be placed into service. Whenever your application needs to scale out, Amazon EC2 Auto Scaling uses the pre-initialized instances from the warm pool rather than launching cold instances. This allows for any final initialization process to run before the instance is placed into service. For more information, see Configuring pre-initialized instances for your Amazon ECS Auto Scaling group.

For more information about creating an Amazon EC2 Auto Scaling launch template, see Auto Scaling launch templates in the *Amazon EC2 Auto Scaling User Guide*. For more information about creating an Amazon EC2 Auto Scaling group, see Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.

Amazon EC2 container instance security considerations for Amazon ECS

You should consider a single container instance and its access within your threat model. For example, a single affected task might be able to leverage the IAM permissions of a non-infected task on the same instance.

We recommend that you use the following to help prevent this:

- Do not use administrator privileges when running your tasks.
- Assign a task role with least-privileged access to your tasks.

  The container agent automatically creates a token with a unique credential ID which are used to access Amazon ECS resources.
- To prevent containers run by tasks that use the `awsvpc` network mode from accessing the credential information supplied to the Amazon EC2 instance profile, while still allowing the permissions that are provided by the task role set the `ECS_AWSVPC_BLOCK_IMDS` agent configuration variable to true in the agent configuration file and restart the agent.
- Use Amazon GuardDuty Runtime Monitoring to detect threats for clusters and containers within your AWS environment. Runtime Monitoring uses a GuardDuty security agent that adds runtime visibility into individual Amazon ECS workloads, for example, file access, process execution, and network connections. For more information, see GuardDuty Runtime Monitoring in the *GuardDuty User Guide*.

Creating an Amazon ECS cluster for the Amazon EC2 launch type

You create a cluster to define the infrastructure your tasks and services run on.

Before you begin, be sure that you've completed the steps in Set up to use Amazon ECS and assign the appropriate IAM permission. For more information, see the section called “Amazon ECS cluster examples”. The Amazon ECS console provides a simple way to create the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack.

To make the cluster creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

You can register Amazon EC2 instances when you create the cluster or register additional instances with the cluster after it has been created.

You can modify the following default options:

- Change the subnets where your instances launch.
- Change the security groups used to control traffic to the container instances.
- Add a namespace to the cluster.

  A namespace allows services that you create in the cluster can connect to the other services in the namespace without additional configuration. For more information, see Interconnect Amazon ECS services.
- Assign a AWS KMS key for your managed storage. For information about how to create a key, see Create a KMS key in the *AWS Key Management Service User Guide*.
- Assign a AWS KMS key for your Fargate ephemeral storage. For information about how to create a key, see Create a KMS key in the *AWS Key Management Service User Guide*.
- Configure the AWS KMS key and logging for ECS Exec.
- Add tags to help you identify your cluster.

Auto Scaling group options

When you use Amazon EC2 instances, you must specify an Auto Scaling group to manage the infrastructure that your tasks and services run on.

When you choose to create a new Auto Scaling group, it is automatically configured for the following behavior:

- Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group.
- Amazon ECS will not prevent Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. For more information, see Instance Protection in the *AWS Auto Scaling User Guide*.

You configure the following Auto Scaling group properties which determine the type and number of instances to launch for the group:

- The Amazon ECS-optimized AMI.
- The instance type.
- The SSH key pair that proves your identity when you connect to the instance. For information about how to create SSH keys, see [Amazon EC2 key pairs and Linux instances](#) in the *Amazon EC2 User Guide*.
- The minimum number of instances to launch for the Auto Scaling group.
- The maximum number of instances that are started for the Auto Scaling group.

  In order for the group to scale out, the maximum must be greater than 0.

Amazon ECS creates an Amazon EC2 Auto Scaling launch template and Auto Scaling group on your behalf as part of the AWS CloudFormation stack. The values that you specified for the AMI, the instance types, and the SSH key pair are part of the launch template. The templates are prefixed with `EC2ContainerService-<ClusterName>`, which makes them easy to identify. The Auto Scaling groups are prefixed with `<ClusterName>-ECS-Infra-ECSAutoScalingGroup`.

Instances launched for the Auto Scaling group use the launch template.

Networking options

By default instances are launched into the default subnets for the Region. The security groups, which control the traffic to your container instances, currently associated with the subnets are used. You can changed the subnets and security groups for the instances.

You can choose an existing subnet. You can either use an existing security group, or create a new one, When you create a new security group, you need to specify at least one inbound rule.

The inbound rules determine what traffic can reach your container instances and include the following properties:

- The protocol to allow
- The range of ports to allow
- The inbound traffic (source)

To allow inbound traffic from a specific address or CIDR block, use **Custom** for **Source** with the allowed CIDR.

To allow inbound traffic from all destinations, use **Anywhere** for **Source**. This automatically adds the 0.0.0.0/0 IPv4 CIDR block and ::/0 IPv6 CIDR block.

To allow inbound traffic from your local computer, use **Source group** for **Source**. This automatically adds the current IP address of your local computer as the allowed source.

To create a new cluster (Amazon ECS console)

Before you begin, assign the appropriate IAM permission. For more information, see the section called "Amazon ECS cluster examples".

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, configure the following:
   - For **Cluster name**, enter a unique name.

     The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.
   - (Optional) To have the namespace used for Service Connect be different from the cluster name, for **Namespace**, enter a unique name.
6. Add Amazon EC2 instances to your cluster, expand **Infrastructure** and then select **Amazon EC2 instances**.

   Next, configure the Auto Scaling group which acts as the capacity provider:

   a. To using an existing Auto Scaling group, from **Auto Scaling group (ASG)**, select the group.
   b. To create a Auto Scaling group, from **Auto Scaling group (ASG)**, select **Create new group**, and then provide the following details about the group:
      - For **Provisioning model**, choose whether to use **On-demand** instances or **Spot** Instances.
      - If you choose to use Spot Instances, for **Allocation Strategy**, choose what Spot capacity pools (instance types and Availability Zones) are used for the instances.

For most workloads, you can choose **Price capacity optimized**.

For more information, see Allocation strategies for Spot Instances in the *Amazon EC2 User Guide*.

- For **Container instance Amazon Machine Image (AMI)**, choose the Amazon ECS-optimized AMI for the Auto Scaling group instances.
- For **EC2 instance type**, choose the instance type for your workloads.

  Managed scaling works best if your Auto Scaling group uses the same or similar instance types.
- For **EC2 instance role**, choose an existing container instance role, or you can create a new one.

  For more information, see Amazon ECS container instance IAM role.
- For **Capacity**, enter the minimum number and the maximum number of instances to launch in the Auto Scaling group.
- For **SSH key pair**, choose the pair that proves your identity when you connect to the instance.
- To allow for larger image and storage, for **Root EBS volume size**, enter the value in GiB.

7. (Optional) To change the VPC and subnets, under **Networking for Amazon EC2 instances**, perform any of the following operations:

   - To remove a subnet, under **Subnets**, choose **X** for each subnet that you want to remove.
   - To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, choose the subnets.
   - Choose the security groups. Under **Security group**, choose one of the following options:
     - To use an existing security group, choose **Use an existing security group**, and then choose the security group.
     - To create a security group, choose **Create a new security group**. Then, choose **Add rule** for each inbound rule.

       For information about inbound rules, see Networking options.
   - To automatically assign public IP addresses to your Amazon EC2 container instances, for **Auto-assign public IP**, choose one of the following options:

    - **Use subnet setting** – Assign a public IP address to the instances when the subnet that the instances launch in are a public subnet.
    - **Turn on** – Assign a public IP address to the instances.
8. (Optional) Use Container Insights, expand **Monitoring**, and then choose one of the following options:
    - To use the recommended Container Insights with enhanced observability, choose **Container Insights with enhanced observability**.
    - To use Container Insights, choose **Container Insights**.
9. (Optional) To use ECS Exec to debug tasks in the cluster, expand **Troubleshooting configuration**, and then configure the following:
    - (Optional) For **AWS KMS key for ECS Exec**, enter the ARN of the AWS KMS key you want to use to encrypt the ECS Exec session data.
    - (Optional) For **ECS Exec logging**, choose the log destination:
        - To send logs to CloudWatch Logs, choose **Amazon CloudWatch**.
        - To send logs to Amazon S3, choose **Amazon S3**.
        - To disable logging, choose **None**.
10. (Optional)

    If you use Runtime Monitoring with the manual option and you want to have this cluster monitored by GuardDuty, choose **Add tag** and do the following:
    - For **Key**, enter **`guardDutyRuntimeMonitoringManaged`**
    - For **Value**, enter **`true`**.
11. (Optional) Encrypt the data on managed storage. Under **Encryption**, for **Managed storage**, enter the ARN of the AWS KMS key you want to use to encrypt the managed storage data.
12. (Optional) To manage the cluster tags, expand **Tags**, and then perform one of the following operations:

    [Add a tag] Choose **Add tag** and do the following:
    - For **Key**, enter the key name.
    - For **Value**, enter the key value.

[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

13. Choose **Create**.

Next steps

After you create the cluster, you can create task definitions for your applications and then run them as standalone tasks, or as part of a service. For more information, see the following:

- Amazon ECS task definitions
- Running an application as an Amazon ECS task
- Creating an Amazon ECS rolling update deployment

Automatically manage Amazon ECS capacity with cluster auto scaling

Amazon ECS can manage the scaling of Amazon EC2 instances that are registered to your cluster. This is referred to as Amazon ECS *cluster auto scaling*. You turn on managed scaling when you create the Amazon ECS Auto Scaling group capacity provider. Then, you set a target percentage (the `targetCapacity`) for the instance utilization in this Auto Scaling group. Amazon ECS creates two custom CloudWatch metrics and a target tracking scaling policy for your Auto Scaling group. Amazon ECS then manages the scale-in and scale-out actions based on the resource utilization that your tasks use.

For each Auto Scaling group capacity provider that's associated with a cluster, Amazon ECS creates and manages the following resources:

- A low metric value CloudWatch alarm
- A high metric value CloudWatch alarm
- A target tracking scaling policy

> **Note**
>
> Amazon ECS creates the target tracking scaling policy and attaches it to the Auto Scaling group. To update the target tracking scaling policy, update the capacity provider managed scaling settings, rather than updating the scaling policy directly.

When you turn off managed scaling or disassociate the capacity provider from a cluster, Amazon ECS removes both CloudWatch metrics and the target tracking scaling policy resources.

Amazon ECS uses the following metrics to determine what actions to take:

`CapacityProviderReservation`

The percent of container instances in use for a specific capacity provider. Amazon ECS generates this metric.

Amazon ECS sets the `CapacityProviderReservation` value to a number between 0-100. Amazon ECS uses the following formula to represent the ratio of how much capacity remains in the Auto Scaling group. Then, Amazon ECS publishes the metric to CloudWatch. For more information about how the metric is calculated, see [Deep Dive on Amazon ECS Cluster Auto Scaling].

```
CapacityProviderReservation = (number of instances needed) / (number of running
 instances) x 100
```

`DesiredCapacity`

The amount of capacity for the Auto Scaling group. This metric isn't published to CloudWatch.

Amazon ECS publishes the `CapacityProviderReservation` metric to CloudWatch in the `AWS/ECS/ManagedScaling` namespace. The `CapacityProviderReservation` metric causes one of the following actions to occur:

The `CapacityProviderReservation` value equals `targetCapacity`

The Auto Scaling group doesn't need to scale in or scale out. The target utilization percentage has been reached.

The `CapacityProviderReservation` value is greater than `targetCapacity`

There are more tasks using a higher percentage of the capacity than your `targetCapacity` percentage. The increased value of the `CapacityProviderReservation` metric causes the associated CloudWatch alarm to act. This alarm updates the `DesiredCapacity` value for the Auto Scaling group. The Auto Scaling group uses this value to launch EC2 instances, and then register them with the cluster.

When the `targetCapacity` is the default value of 100 %, the new tasks are in the `PENDING` state during the scale-out because there is no available capacity on the instances to run the tasks. After the new instances register with ECS, these tasks will start on the new instances.

The `CapacityProviderReservation` value is less than `targetCapacity`

There are less tasks using a lower percentage of the capacity than your `targetCapacity` percentage and there is at least one instance that can be terminated. The decreased value of the `CapacityProviderReservation` metric causes the associated CloudWatch alarm to act. This alarm updates the `DesiredCapacity` value for the Auto Scaling group. The Auto Scaling group uses this value to terminate EC2 container instances, and then deregister them from the cluster.

The Auto Scaling group follows the group termination policy to determine which instances it terminates first during scale-in events. Additionally it avoids instances with the instance scale-in protection setting turned on. Cluster auto scaling can manage which instances have the instance scale-in protection setting if you turn on managed termination protection. For more information about managed termination protection, see Control the instances Amazon ECS terminates. For more information about how Auto Scaling groups terminate instances, see Control which Auto Scaling instances terminate during scale in in the *Amazon EC2 Auto Scaling User Guide*.

Consider the following when using cluster auto scaling:

- Don't change or manage the desired capacity for the Auto Scaling group that's associated with a capacity provider with any scaling policies other than the one Amazon ECS manages.
- When Amazon ECS scales out from 0 instances, it automatically launches 2 instances.
- Amazon ECS uses the `AWSServiceRoleForECS` service-linked IAM role for the permissions that it requires to call AWS Auto Scaling on your behalf. For more information, see Using service-linked roles for Amazon ECS.
- When using capacity providers with Auto Scaling groups, the user, group, or role that creates the capacity providers requires the `autoscaling:CreateOrUpdateTags` permission. This is because Amazon ECS adds a tag to the Auto Scaling group when it associates it with the capacity provider.

> **Important**
>
> Make sure any tooling that you use doesn't remove the `AmazonECSManaged` tag from the Auto Scaling group. If this tag is removed, Amazon ECS can't manage the scaling.

- Cluster auto scaling doesn't modify the **MinimumCapacity** or **MaximumCapacity** for the group. For the group to scale out, the value for **MaximumCapacity** must be greater than zero.
- When Auto Scaling (managed scaling) is turned on, a capacity provider can only be connected to one cluster at the same time. If your capacity provider has managed scaling turned off, you can associate it with multiple clusters.
- When managed scaling is turned off, the capacity provider doesn't scale in or scale out. You can use a capacity provider strategy to balance your tasks between capacity providers.
- The `binpack` strategy is the most efficient strategy in terms of capacity.
- When the target capacity is less than 100%, the placement strategy needs to use the `binpack` strategy must have a higher order than the `spread` strategy. This prevents the capacity provider from scaling out until each task has a dedicated instance or the limit is reached.

Turn on cluster auto scaling

You can turn on cluster auto scaling by using the Console or the AWS CLI.

When you create a cluster for the EC2 launch type using the console, Amazon ECS creates an Auto Scaling group on your behalf and sets the target capacity. For more information, see Creating an Amazon ECS cluster for the Amazon EC2 launch type.

You can also create an Auto Scaling group, and then assign it to a cluster. For more information, see Updating an Amazon ECS capacity provider.

When you use the AWS CLI, after you create the cluster

1. Before you create the capacity provider, you need to create an Auto Scaling group. For more information, see Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.
2. Use `put-cluster-capacity-providers` to modify the cluster capacity provider. For more information, see Turning on Amazon ECS cluster auto scaling.

Optimize Amazon ECS cluster auto scaling

Customers who run Amazon ECS on Amazon EC2 can take advantage of cluster auto scaling to manage the scaling of Amazon EC2 Auto Scaling groups. With cluster auto scaling, you can configure Amazon ECS to scale your Auto Scaling group automatically, and just focus on running your tasks. Amazon ECS ensures the Auto Scaling group scales in and out as needed with no further intervention required. Amazon ECS capacity providers are used to manage the infrastructure in your cluster by ensuring there are enough container instances to meet the demands of your application. To learn how cluster auto scaling works under the hood, see Deep Dive on Amazon ECS Cluster Auto Scaling.

Cluster auto scaling relies on a CloudWatch based integration with Auto Scaling group for adjusting cluster capacity. Therefore it has inherent latency associated with

- Publishing the CloudWatch metrics,
- The time taken for the metric `CapacityProviderReservation` to breach CloudWatch alarms (both high and low)
- The time taken by a newly launched Amazon EC2 instance to warm-up. You can take the following actions to make cluster auto scaling more responsive for faster deployments:

Capacity provider step scaling sizes

Amazon ECS capacity providers will grow/shrink the container instances to meet the demands of your application. The minimum number of instances that Amazon ECS will launch is set to 1 by default. This may add additional time to your deployments, if several instances are required for placing your pending tasks. You can increase the `minimumScalingStepSize` via the Amazon ECS API to increase the minimum number of instances that Amazon ECS scales in or out at a time. A `maximumScalingStepSize` that is too low can limit how many container instances are scaled in or out at a time, which can slow down your deployments.

> **Note**
>
> This configuration is currently only available via the `CreateCapacityProvider` or `UpdateCapacityProvider` APIs.

Instance warm-up period

The instance warm-up period is the period of time after which a newly launched Amazon EC2 instance can contribute to CloudWatch metrics for the Auto Scaling group. After the specified warm-up period expires, the instance is counted toward the aggregated metrics of the Auto Scaling group, and cluster auto scaling proceeds with its next iteration of calculations to estimate the number instances required.

The default value for `instanceWarmupPeriod` is 300 seconds, which you can configure to a lower value via the `CreateCapacityProvider` or `UpdateCapacityProvider` APIs for more responsive scaling. We recommend that you set the value to greater than 60 seconds so that you can avoid over-provisioning.

Spare capacity

If your capacity provider has no container instances available for placing tasks, then it needs to increase (scale out) cluster capacity by launching Amazon EC2 instances on the fly, and wait for them to boot up before it can launch containers on them. This can significantly lower the task launch rate. You have two options here.

In this case, having spare Amazon EC2 capacity already launched and ready to run tasks will increase the effective task launch rate. You can use the `Target Capacity` configuration to indicate that you wish to maintain spare capacity in your clusters. For example, by setting `Target Capacity` at 80%, you indicate that your cluster needs 20% spare capacity at all times. This spare capacity can allow any standalone tasks to be immediately launched, ensuring task launches are not throttled. The trade-off for this approach is potential increased costs of keeping spare cluster capacity.

An alternate approach you can consider is adding headroom to your service, not to the capacity provider. This means that instead of reducing `Target Capacity` configuration to launch spare capacity, you can increase the number of replicas in your service by modifying the target tracking scaling metric or the step scaling thresholds of the service auto scaling. Note that this approach will only be helpful for spiky workloads, but won't have an effect when you’re deploying new services and going from 0 to N tasks for the first time. For more information about the related scaling policies, see Target Tracking Scaling Policies or Step Scaling Policies in the *Amazon Elastic Container Service Developer Guide*.

Amazon ECS managed scaling behavior

When you have Auto Scaling group capacity providers that use managed scaling, Amazon ECS estimates the optimal number of instances to add to your cluster and uses the value to determine how many instances to request or release.

Managed scale-out behavior

Amazon ECS selects a capacity provider for each task by following the capacity provider strategy from the service, standalone task, or the cluster default. Amazon ECS follows the rest of these steps for a single capacity provider.

Tasks without a capacity provider strategy are ignored by capacity providers. A pending task that doesn't have a capacity provider strategy won't cause any capacity provider to scale out. Tasks or services can't set a capacity provider strategy if that task or service sets a launch type.

The following describes the scale-out behavior in more detail.

- Group all of the provisioning tasks for this capacity provider so that each group has the same exact resource requirements.
- When you use multiple instance types in an Auto Scaling group, the instance types in the Auto Scaling group are sorted by their parameters. These parameters include vCPU, memory, elastic network interfaces (ENIs), ports, and GPUs. The smallest and the largest instance types for each parameter are selected. For more information about how to choose the instance type, see Amazon EC2 container instances for Amazon ECS.

> **Important**
>
> If a group of tasks have resource requirements that are greater than the smallest instance type in the Auto Scaling group, then that group of tasks can’t run with this capacity provider. The capacity provider doesn’t scale the Auto Scaling group. The tasks remain in the `PROVISIONING` state.
> To prevent tasks from staying in the `PROVISIONING` state, we recommend that you create separate Auto Scaling groups and capacity providers for different minimum resource requirements. When you run tasks or create services, only add capacity providers to the capacity provider strategy that can run the task on the smallest instance type in the Auto Scaling group. For other parameters, you can use placement constraints

- For each group of tasks, Amazon ECS calculates the number of instances that are required to run the unplaced tasks. This calculation uses a `binpack` strategy. This strategy accounts for the vCPU, memory, elastic network interfaces (ENI), ports, and GPUs requirements of the tasks. It also accounts for the resource availability of the Amazon EC2 instances. The values for the largest instance types are treated as the maximum calculated instance count. The values for the smallest instance type are used as protection. If the smallest instance type can't run at least one instance of the task, the calculation considers the task as not compatible. As a result, the task is excluded from scale-out calculation. When all the tasks aren't compatible with the smallest instance type, cluster auto scaling stops and the `CapacityProviderReservation` value remains at the `targetCapacity` value.
- Amazon ECS publishes the `CapacityProviderReservation` metric to CloudWatch with respect to the `minimumScalingStepSize` if either of the following is the case.
  - The maximum calculated instance count is less than the minimum scaling step size.
  - The lower value of either the `maximumScalingStepSize` or the maximum calculated instance count.
- CloudWatch alarms use the `CapacityProviderReservation` metric for capacity providers. When the `CapacityProviderReservation` metric is greater than the `targetCapacity` value, alarms also increase the `DesiredCapacity` of the Auto Scaling group. The `targetCapacity` value is a capacity provider setting that's sent to the CloudWatch alarm during the cluster auto scaling activation phase.

  The default `targetCapacity` is 100%.
- The Auto Scaling group launches additional EC2 instances. To prevent over-provisioning, Auto Scaling makes sure that recently launched EC2 instance capacity is stabilized before it launches new instances. Auto Scaling checks if all existing instances have passed the `instanceWarmupPeriod` (now minus the instance launch time). The scale-out is blocked for instances that are within the `instanceWarmupPeriod`.

  The default number of seconds for a newly launched instance to warm up is 300.

For more information, see [Deep dive on Amazon ECS cluster auto scaling](#).

Scale-out considerations

Consider the following for the scale-out process:

- Although there are multiple placement constraints, we recommend that you only use the `distinctInstance` task placement constraint. This prevents the scale-out process from stopping because you're using a placement constraint that's not compatible with the sampled instances.
- Managed scaling works best if your Auto Scaling group uses the same or similar instance types.
- When a scale-out process is required and there are no currently running container instances, Amazon ECS always scales-out to two instances initially, and then performs additional scale-out or scale-in processes. Any additional scale-out waits for the instance warmup period. For scale-in processes, Amazon ECS waits 15 minutes after a scale-out process before starting scale-in processes at all times.
- The second scale-out step needs to wait until the `instanceWarmupPeriod` expires, which might affect the overall scale limit. If you need to reduce this time, make sure that `instanceWarmupPeriod` is large enough for the EC2 instance to launch and start the Amazon ECS agent (which prevents over provisioning).
- Cluster auto scaling supports Launch Configuration, Launch Templates, and multiple instance types in the capacity provider Auto Scaling group. You can also use attribute-based instance type selection without multiple instances types.
- When using an Auto Scaling group with On-Demand instances and multiple instance types or Spot Instances, place the larger instance types higher in the priority list and don't specify a weight. Specifying a weight isn't supported at this time. For more information, see Auto Scaling groups with multiple instance types in the *AWS Auto Scaling User Guide*.
- Amazon ECS then launch either the `minimumScalingStepSize`, if the maximum calculated instance count is less than the minimum scaling step size, or the lower of either the `maximumScalingStepSize` or the maximum calculated instance count value.
- If an Amazon ECS service or `run-task` launches a task and the capacity provider container instances don't have enough resources to start the task, then Amazon ECS limits the number of tasks with this status for each cluster and prevents any tasks from exceeding this limit. For more information, see *Service quotas*.

Managed scale-in behavior

Amazon ECS monitors container instances for each capacity provider within a cluster. When a container instance isn't running any tasks, the container instance is considered empty and Amazon ECS starts the scale-in process.

CloudWatch scale-in alarms require 15 data points (15 minutes) before the scale-in process for the Auto Scaling group starts. After the scale-in process starts until Amazon ECS needs to reduce the number of registered container instances, the Auto Scaling group sets the `DesireCapacity` value to be greater than one instance and less than 50% each minute.

When Amazon ECS requests a scale-out (when `CapacityProviderReservation` is greater than 100) while a scale-in process is in progress, the scale-in process is stopped and starts from the beginning if required.

The following describes the scale-in behavior in more detail:

1. Amazon ECS calculates the number of container instances that are empty. A container instance is considered empty even when daemon tasks are running.
2. Amazon ECS sets the `CapacityProviderReservation` value to a number between 0-100 that uses the following formula to represent the ratio of how big the Auto Scaling group needs to be relative to how big it actually is, expressed as a percentage. Then, Amazon ECS publishes the metric to CloudWatch. For more information about how the metric is calculated, see [Deep Dive on Amazon ECS Cluster Auto Scaling](#)

   ```
   CapacityProviderReservation = (number of instances needed) / (number of running
    instances) x 100
   ```
3. The `CapacityProviderReservation` metric generates a CloudWatch alarm. This alarm updates the `DesiredCapacity` value for the Auto Scaling group. Then, one of the following actions occurs:
   - If you don't use capacity provider managed termination, the Auto Scaling group selects EC2 instances using the Auto Scaling group termination policy and terminates the instances until the number of EC2 instances reaches the `DesiredCapacity`. The container instances are then deregistered from the cluster.
   - If all the container instances use managed termination protection, Amazon ECS removes the scale-in protection on the container instances that are empty. The Auto Scaling group will then be able to terminate the EC2 instances. The container instances are then deregistered from the cluster.

Control the instances Amazon ECS terminates

> **Important**
>
> You must turn on Auto Scaling *instance scale-in protection* on the Auto Scaling group to use the managed termination protection feature of cluster auto scaling.

Managed termination protection allows cluster auto scaling to control which instances are terminated. When you used managed termination protection, Amazon ECS only terminates EC2 instances that don't have any running Amazon ECS tasks. Tasks that are run by a service that uses the `DAEMON` scheduling strategy are ignored and an instance can be terminated by cluster auto scaling even when the instance is running these tasks. This is because all of the instances in the cluster are running these tasks.

Amazon ECS first turns on the *instance scale-in protection* option for the EC2 instances in the Auto Scaling group. Then, Amazon ECS places the tasks on the instances. When all non-daemon tasks are stopped on an instance, Amazon ECS initiates the scale-in process and turns off scale-in protection for the EC2 instance. The Auto Scaling group can then terminate the instance.

Auto Scaling *instance scale-in protection* controls which EC2 instances can be terminated by Auto Scaling. Instances with the scale-in feature turned on can't be terminated during the scale-in process. For more information about Auto Scaling instance scale-in protection, see Using instance scale-in protection in the *Amazon EC2 Auto Scaling User Guide*.

You can set the `targetCapacity` percentage so that you have spare capacity. This helps future tasks launch more quickly because the Auto Scaling group does not have to launch more instances. Amazon ECS uses the target capacity value to manage the CloudWatch metric that the service creates. Amazon ECS manages the CloudWatch metric. The Auto Scaling group is treated as a steady state so that no scaling action is required. The values can be from 0-100%. For example, to configure Amazon ECS to keep 10% free capacity on top of that used by Amazon ECS tasks, set the target capacity value to 90%. Consider the following when setting the `targetCapacity` value on a capacity provider.

- A `targetCapacity` value of less than 100% represents the amount of free capacity (Amazon EC2 instances) that need to be present in the cluster. Free capacity means that there are no running tasks.

- Placement constraints such as Availability Zones, without additional `binpack` forces Amazon ECS to eventually run one task for each instance, which might not be the desired behavior.

You must turn on Auto Scaling instance scale-in protection on the Auto Scaling group to use managed termination protection. If you don't turn on scale-in protection, then turning on managed termination protection can lead to undesirable behavior. For example, you may have instances stuck in draining state. For more information, see Using instance scale-in protection in the *Amazon EC2 Auto Scaling User Guide*.

When you use termination protection with a capacity provider, don't perform any manual actions, like detaching the instance, on the Auto Scaling group associated with the capacity provider. Manual actions can break the scale-in operation of the capacity provider. If you detach an instance from the Auto Scaling group, you need to also deregister the detached instance from the Amazon ECS cluster.

Updating managed termination protection for Amazon ECS capacity providers

When you use managed termination protection, you need to update the setting for existing capacity providers.

Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster page, chooset the **Infrastructure** tab.
4. Choose the capacity provider.
5. Choose **Update** to modify the capacity provider settings.
6. Under **Auto Scaling group settings**, toggle **Managed termination protection** to enable or disable the feature.
7. Choose **Update**.

AWS CLI

You can update a capacity provider's managed termination protection setting using the `update-capacity-provider` command:

To enable managed termination protection:

```
aws ecs update-capacity-provider \
  --capacity-provider CapacityProviderName \
  --auto-scaling-group-provider
 "managedScaling={status=ENABLED,targetCapacity=70,minimumScalingStepSize=1,maximumScalingStepS
```

To disable managed termination protection:

```
aws ecs update-capacity-provider \
  --capacity-provider CapacityProviderName \
  --auto-scaling-group-provider
 "managedScaling={status=ENABLED,targetCapacity=70,minimumScalingStepSize=1,maximumScalingStepS
```

Note

It might take a few minutes for the changes to take effect across your cluster. When enabling managed termination protection, instances that are already running tasks will be protected from scale-in events. When disabling managed termination protection, the protection flag will be removed from instances during the next ECS capacity provider management cycle.

Console for running tasks

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster page, chooset the **Tasks** tab.
4. Choose the task.
5. Under **Configuration**, toggle **Managed termination protection** to enable or disable the feature.
6. Choose **Configure task scale-in protection**.

   The **Configure task scale-in protection** dialog box displays

   a. Under **Task scale-in protection**, toggle **Turn on**.

   b. For **Expires in minutes**, enter the number of minutes before task scale-in protection ends.

   c. Choose **Update**

Turning on Amazon ECS cluster auto scaling

You turn on cluster auto scaling so that Amazon ECS manages the scaling of Amazon EC2 instances that are registered to your cluster.

If you want to use the console to turn on Cluster auto scaling, see see Creating a capacity provider for Amazon ECS.

Before you begin, create an Auto Scaling group and a capacity provider. For more information, see the section called "Capacity providers for the EC2 launch type".

To turn on cluster auto scaling, you associate the capacity provider with the cluster, Then you turn on cluster auto scaling.

1. Use the `put-cluster-capacity-providers` command to associate one or more capacity providers with the cluster.

   To add the AWS Fargate capacity providers, include the `FARGATE` and `FARGATE_SPOT` capacity providers in the request. For more information, see `put-cluster-capacity-providers` in the *AWS CLI Command Reference*.

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
     --capacity-providers CapacityProviderName FARGATE FARGATE_SPOT \
     --default-capacity-provider-strategy capacityProvider=CapacityProvider,weight=1
   ```

   To add an Auto Scaling group for the EC2 launch type, include the Auto Scaling group name in the request. For more information, see `put-cluster-capacity-providers` in the *AWS CLI Command Reference*.

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
     --capacity-providers CapacityProviderName \
     --default-capacity-provider-strategy capacityProvider=CapacityProvider,weight=1
   ```

2. Use the `describe-clusters` command to verify that the association was successful. For more information, see `describe-clusters` in the *AWS CLI Command Reference*.

   ```
   aws ecs describe-clusters \
     --cluster ClusterName \
   ```

```
  --include ATTACHMENTS
```

3. Use the `update-capacity-provider` command to turn on managed auto scaling for the capacity provider. For more information, see [update-capacity-provider]() in the *AWS CLI Command Reference*.

   ```
   aws ecs update-capacity-provider \
     --capacity-providers CapacityProviderName \
     --auto-scaling-group-provider managedScaling=ENABLED
   ```

Turning off Amazon ECS cluster auto scaling

You turn off cluster auto scaling when you need more granular control of the EC2 instances that are registered to your cluster,

To turn off cluster auto scaling for a cluster, you can either disassociate the capacity provider with managed scaling turned on from the cluster or update the capacity provider to turn off managed scaling.

Disassociate the capacity provider

Use the following steps to disassociate a capacity provider with a cluster.

1. Use the `put-cluster-capacity-providers` command to disassociate the Auto Scaling group capacity provider with the cluster. The cluster can keep the association with the AWS Fargate capacity providers. For more information, see [put-cluster-capacity-providers]() in the *AWS CLI Command Reference*.

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
     --capacity-providers FARGATE FARGATE_SPOT \
     --default-capacity-provider-strategy '[]'
   ```

   Use the `put-cluster-capacity-providers` command to disassociate the Auto Scaling group capacity provider with the cluster. For more information, see [put-cluster-capacity-providers]() in the *AWS CLI Command Reference*.

   ```
   aws ecs put-cluster-capacity-providers \
     --cluster ClusterName \
   ```

```
  --capacity-providers [] \
  --default-capacity-provider-strategy '[]'
```

2. Use the `describe-clusters` command to verify that the disassociation was successful. For more information, see describe-clusters in the *AWS CLI Command Reference*.

```
aws ecs describe-clusters \
  --cluster ClusterName \
  --include ATTACHMENTS
```

Turn off managed scaling for the capacity provider

Use the following steps to turn off managed scaling for the capacity provider.

- Use the `update-capacity-provider` command to turn off managed auto scaling for the capacity provider. For more information, see update-capacity-provider in the *AWS CLI Command Reference*.

```
aws ecs update-capacity-provider \
  --capacity-providers CapacityProviderName \
  --auto-scaling-group-provider managedScaling=DISABLED
```

Creating a capacity provider for Amazon ECS

After the cluster creation completes, you can create a new capacity provider (Auto Scaling group) for the EC2 launch type. Capacity providers help to manage and scale your the infrastructure for your applications.

Before you create the capacity provider, you need to create an Auto Scaling group. For more information, see Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.

To create a capacity provider for the cluster (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : *name*** page, choose **Infrastructure**, and then choose **Create**.
5. On the **Create capacity providers** page, configure the following options.

   a. Under **Basic details**, for **Capacity provider name**, enter a unique capacity provider name.
   b. Under **Auto Scaling group**, for **Use an existing Auto Scaling group**, choose the Auto Scaling group.
   c. (Optional) To configure a scaling policy, under **Scaling policies**, configure the following options.

      - To have Amazon ECS manage the scale-in and scale-out actions, select **Turn on managed scaling**.
      - To prevent EC2 instance with running Amazon ECS tasks from being terminated, select **Turn on scaling protection**.
      - For **Set target capacity**, enter the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy.

6. Choose **Create**.

Updating an Amazon ECS capacity provider

When you use an Auto Scaling group as a capacity provider, you can modify the group's scaling policy.

To update a capacity provider for the cluster (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : *`name`*** page, choose **Infrastructure**, and then choose **Update**.
5. On the **Create capacity providers** page, configure the following options.

   - Under **Auto Scaling group**, under **Scaling policies**, configure the following options.

     - To have Amazon ECS manage the scale-in and scale-out actions, select **Turn on managed scaling**.
     - To prevent EC2 instances with running Amazon ECS tasks from being terminated, select **Turn on scaling protection**.
     - For **Set target capacity**, enter the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy.

6. Choose **Update**.

Deleting an Amazon ECS capacity provider

If you are finished using an Auto Scaling group capacity provider, you can delete it. After the group is deleted, the Auto Scaling group capacity provider transitions to the `INACTIVE` state. Capacity providers with an `INACTIVE` status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so you should not rely on `INACTIVE` capacity providers persisting. Before the Auto Scaling group capacity provider is deleted, the capacity provider must be removed from the capacity provider strategy from all services. You can use the `UpdateService` API or the update service workflow in the Amazon ECS console to remove a capacity provider from a service's capacity provider strategy. Use the **Force new deployment** option to ensure that any tasks using the Amazon EC2 instance capacity provided by the capacity provider are transitioned to use the capacity from the remaining capacity providers.

To delete a capacity provider for the cluster (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster :** ***`name`*** page, choose **Infrastructure**, the Auto Scaling group, and then choose **Delete**.
5. In the confirmation box, enter **delete** ***`Auto Scaling group name`***
6. Choose **Delete**.

Safely stop Amazon ECS workloads running on EC2 instances

Managed instance draining facilitates graceful termination of Amazon EC2 instances. This allows your workloads to stop safely and be rescheduled to non-terminating instances. Infrastructure maintenance and updates are performed without worrying about disruption to workloads. By using managed instance draining, you simplify your infrastructure management workflows that require replacement of Amazon EC2 instances while you ensure resilience and availability of your applications.

Amazon ECS managed instance draining works with Auto Scaling group instance replacements. Based on instance refresh and maximum instance lifetime, customers can ensure that they stay compliant with the latest OS and security mandates for their capacity.

Managed instance draining can only be used with Amazon ECS capacity providers. You can turn on managed instance draining when you create or update your Auto Scaling group capacity providers using the Amazon ECS console, AWS CLI, or SDK.

The following events are covered by Amazon ECS managed instance draining.

- Auto Scaling group instance refresh - Use instance refresh to perform rolling replacement of your Amazon EC2 instances in your Auto Scaling group instead of manually doing it in batches. This is useful when you need to replace a large number of instances. An instance refresh is initiated through the Amazon EC2 console or the `StartInstanceRefresh` API. Make sure you select `Replace` for Scale-in protection when calling `StartInstanceRefresh` if you're using managed termination protection.
- Maximum instance lifetime - You can define a maximum lifetime when it comes to replacing Auto Scaling group instances. This is helpful for scheduling replacement instances based on internal security policies or compliance.
- Auto Scaling group scale-in - Based on scaling policies and scheduled scaling actions, Auto Scaling group supports automatic scaling of instances. By using an Auto Scaling group as an Amazon ECS capacity provider, you can scale-in Auto Scaling group instances when no tasks are running in them.
- Auto Scaling group health checks - Auto Scaling group supports many health checks to manage termination of unhealthy instances.
- AWS CloudFormation stack updates - You can add an `UpdatePolicy` attribute to your AWS CloudFormation stack to perform rolling updates when group changes.
- Spot capacity rebalancing - The Auto Scaling group tries to proactively replace Spot Instances that have a higher risk of interruption based on Amazon EC2 capacity rebalance notice. The Auto Scaling group terminates the old instance when the replacement is launched and healthy. Amazon ECS managed instance draining drains the Spot Instance the same way it drains a non-Spot Instance.
- Spot interruption - Spot Instances are terminated with a two minute notice. Amazon ECS-managed instance draining puts the instance in draining state in response.

Amazon EC2 Auto Scaling lifecycle hooks with managed instance draining

Auto Scaling group lifecycle hooks enable customer to create solutions that are triggered by certain events in the instance lifecycle and perform a custom action when that certain event occurs. An Auto Scaling group allows for up to 50 hooks. Multiple termination hooks can exist and are performed in parallel, and Auto Scaling group waits for all hooks to finish before terminating an instance.

In addition to the Amazon ECS-managed hook termination, you can also configure your own lifecycle termination hooks. Lifecycle hooks have a `default action`, and we recommend setting `continue` as the default to ensure other hooks, such as the Amazon ECS managed hook, aren't impacted by any errors from custom hooks.

If you've already configured an Auto Scaling group termination lifecycle hook and also enabled Amazon ECS managed instance draining, both lifecycle hooks are performed. The relative timings, however, are not guaranteed. Lifecycle hooks have a `default action` setting to specify the action to take when timeout elapses. In case of failures we recommend using `continue` as the default result in your custom hook. This ensures other hooks, particularly the Amazon ECS managed hooks, aren't impacted by any errors in your custom lifecycle hook. The alternative result of `abandon` causes all other hooks to be skipped and should be avoided. For more information about Auto Scaling group lifecycle hooks see Amazon EC2 Auto Scaling lifecycle hooks in the *Amazon EC2* Auto Scaling User Guide.

Tasks and managed instance draining

Amazon ECS managed instance draining uses the existing draining feature found in container instances. The container instance draining feature performs replacement and stops for replica tasks that belong to an Amazon ECS service. A standalone task, like one invoked by `RunTask`, that is in the `PENDING` or `RUNNING` state remain unaffected. You have to wait for these to either complete or stop them manually. The container instance remains in the `DRAINING` state until either all tasks are stopped or 48 hours has passed. Daemon tasks are the last to stop after all replica tasks have stopped.

Managed instance draining and managed termination protection

Managed instance draining works even if managed termination is disabled. For information about managed termination protection, see Control the instances Amazon ECS terminates.

The following table summarizes the behavior for different combinations of managed termination and managed draining.

Managed termination	Managed draining	Outcome
Enabled	Enabled	Amazon ECS protects Amazon EC2 instances that are running tasks from being terminated by scale-in events. Any instances undergoing termination, such as those that don't have termination protection set, have received Spot interruption, or are forced by instance refresh are gracefully drained.

Managed termination	Managed draining	Outcome
Disabled	Enabled	Amazon ECS doesn't protect Amazon EC2 instances running tasks from being scaled-in. However, any instances that are being terminated are gracefully drained.

Managed termination	Managed draining	Outcome
Enabled	Disabled	Amazon ECS protects Amazon EC2 instances that are running tasks from being terminated by scale-in events. However, instances can still get terminated by Spot interruption or forced instance refresh, or if they aren't running any tasks. Amazon ECS doesn't perform graceful draining

Managed termination	Managed draining	Outcome
		for these instances, and launches replacement service tasks after they stop.
Disabled	Disabled	Amazon EC2 instances can be scaled-in or terminated at any time, even if they are running Amazon ECS tasks. Amazon ECS will launch replacement service tasks after they stop.

Managed instance draining and Spot Instance draining

With Spot Instance draining, you can set an environment variable `ECS_ENABLE_SPOT_INSTANCE_DRAINING` on the Amazon ECS agent which enables Amazon ECS to place an instance in the draining status in response to the two-minute Spot interruption.

Amazon ECS managed instance draining facilitates graceful shutdown of Amazon EC2 instances undergoing termination due to many reasons, not just Spot interruption. For instance, you can use Amazon EC2 Auto Scaling capacity rebalancing to proactively replace Spot Instance at elevated risk of interruption, and managed instance draining performs graceful shutdown of Spot Instance being replaced. When you use managed instance draining, you don't need to enable Spot instance draining separately, so `ECS_ENABLE_SPOT_INSTANCE_DRAINING` in Auto Scaling group user data is redundant. For more information about Spot Instance draining, see [Spot Instances](#).

How managed instance draining works with EventBridge

Amazon ECS managed instance draining events are published to Amazon EventBridge, and Amazon ECS creates an EventBridge managed rule in your account’s default bus to support managed instance draining. You can filter these events to other AWS services like Lambda, Amazon SNS, and Amazon SQS to monitor and troubleshoot.

- Amazon EC2 Auto Scaling sends an event to EventBridge when a lifecycle hook is invoked.
- Spot interruption notices are published to EventBridge.
- Amazon ECS generates error messages that you can retrieve through the Amazon ECS console and APIs.
- EventBridge has retry mechanisms built in as mitigations for temporary failures.

Configuring Amazon ECS capacity providers to safely shut down instances

You can turn on managed instance draining when you create or update your Auto Scaling group capacity providers using the Amazon ECS console and AWS CLI.

> **Note**
>
> Managed instance draining is on by default when you create a capacity provider.

The following are examples using the AWS CLI for creating a capacity provider with managed instance draining enabled and enabling managed instance draining for a cluster's existing capacity provider.

Create a capacity provider with managed instance draining enabled

To create a capacity provider with managed instance draining enabled, use the `create-capacity-provider` command. Set the `managedDraining` parameter to `ENABLED`.

```
aws ecs create-capacity-provider \
--name capacity-provider \
--auto-scaling-group-provider '{
  "autoScalingGroupArn": "asg-arn",
  "managedScaling": {
    "status": "ENABLED",
    "targetCapacity": 100,
    "minimumScalingStepSize": 1,
    "maximumScalingStepSize": 1
  },
  "managedDraining": "ENABLED",
  "managedTerminationProtection": "ENABLED",
}'
```

Response:

```
{
    "capacityProvider": {
        "capacityProviderArn": "capacity-provider-arn",
        "name": "capacity-provider",
        "status": "ACTIVE",
        "autoScalingGroupProvider": {
            "autoScalingGroupArn": "asg-arn",
            "managedScaling": {
                "status": "ENABLED",
                "targetCapacity": 100,
                "minimumScalingStepSize": 1,
                "maximumScalingStepSize": 1
            },
            "managedTerminationProtection": "ENABLED"
            "managedDraining": "ENABLED"
        }
    }
}
```

Enable managed instance draining for a cluster's existing capacity provider

Enable managed instance draining for a cluster's existing capacity provider uses the `update-capacity-provider` command. You see that `managedDraining` currently says `DISABLED` and `updateStatus` says `UPDATE_IN_PROGRESS`.

```
aws ecs update-capacity-provider \
```

```
--name cp-draining \
--auto-scaling-group-provider '{
  "managedDraining": "ENABLED"
}
```

Response:

```
{
    "capacityProvider": {
        "capacityProviderArn": "cp-draining-arn",
        "name": "cp-draining",
        "status": "ACTIVE",
        "autoScalingGroupProvider": {
            "autoScalingGroupArn": "asg-draining-arn",
            "managedScaling": {
                "status": "ENABLED",
                "targetCapacity": 100,
                "minimumScalingStepSize": 1,
                "maximumScalingStepSize": 1,
                "instanceWarmupPeriod": 300
            },
            "managedTerminationProtection": "DISABLED",
            "managedDraining": "DISABLED" // before update
        },
        "updateStatus": "UPDATE_IN_PROGRESS", // in progress and need describe again to
 find out the result
        "tags": [
        ]
    }
}
```

Use the `describe-clusters` command and include ATTACHMENTS. The `status` of the managed instance draining attachment is PRECREATED, and the overall `attachmentsStatus` is UPDATING.

```
aws ecs describe-clusters --clusters cluster-name --include ATTACHMENTS
```

Response:

```
{
    "clusters": [
        {
```

```
            ...

            "capacityProviders": [
                "cp-draining"
            ],
            "defaultCapacityProviderStrategy": [],
            "attachments": [
                # new precreated managed draining attachment
                {
                    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                    "type": "managed_draining",
                    "status": "PRECREATED",
                    "details": [
                        {
                            "name": "capacityProviderName",
                            "value": "cp-draining"
                        },
                        {
                            "name": "autoScalingLifecycleHookName",
                            "value": "ecs-managed-draining-termination-hook"
                        }
                    ]
                },

                ...

            ],
            "attachmentsStatus": "UPDATING"
        }
    ],
    "failures": []
}
```

When the update is finished, use `describe-capacity-providers`, and you see `managedDraining` is now `ENABLED`.

```
aws ecs describe-capacity-providers --capacity-providers cp-draining
```

Response:

```
{
    "capacityProviders": [
        {
```

```
            "capacityProviderArn": "cp-draining-arn",
            "name": "cp-draining",
            "status": "ACTIVE",
            "autoScalingGroupProvider": {
                "autoScalingGroupArn": "asg-draning-arn",
                "managedScaling": {
                    "status": "ENABLED",
                    "targetCapacity": 100,
                    "minimumScalingStepSize": 1,
                    "maximumScalingStepSize": 1,
                    "instanceWarmupPeriod": 300
                },
                "managedTerminationProtection": "DISABLED",
                "managedDraining": "ENABLED" // successfully update
            },
            "updateStatus": "UPDATE_COMPLETE",
            "tags": []
        }
    ]
}
```

Amazon ECS Managed instance draining troubleshooting

You might need to troubleshoot issues with managed instance draining. The following is an example of an issue and resolution you may come across while using it.

Instances don't terminate after exceeding maximum instance lifetime when using auto scaling.

If your instances aren't terminating even after reaching and exceeding the maximum instance lifetime while using an auto scaling group, it may be because they're protected from scale-in. You can turn off managed termination and allow managed draining to handle instance recycling.

Creating resources for Amazon ECS cluster auto scaling using the AWS Management Console

Learn how to create the resources for cluster auto scaling using the AWS Management Console. Where resources require a name, we use the prefix `ConsoleTutorial` to ensure they all have unique names and to make them easy to locate.

Topics

- Prerequisites

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- The Amazon ECS container instance IAM role is created. For more information, see Amazon ECS container instance IAM role.
- The Amazon ECS service-linked IAM role is created. For more information, see Using service-linked roles for Amazon ECS.
- The Auto Scaling service-linked IAM role is created. For more information, see Service-Linked Roles for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.
- You have a VPC and security group created to use. For more information, see the section called "Create a virtual private cloud".

Step 1: Create an Amazon ECS cluster

Use the following steps to create an Amazon ECS cluster.

Amazon ECS creates an Amazon EC2 Auto Scaling launch template and Auto Scaling group on your behalf as part of the AWS CloudFormation stack.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**, and then choose **Create cluster**.
3. Under **Cluster configuration**, for **Cluster name**, enter `ConsoleTutorial-cluster`.
4. Under **Infrastructure**, clear AWS Fargate (serverless), and then select **Amazon EC2 instances**. Next, configure the Auto Scaling group which acts as the capacity provider.

- Under **Auto Scaling group (ASG)** . Select **Create new ASG**, and then provide the following details about the group:
  - For **Operating system/Architecture**, choose **Amazon Linux 2**.
  - For **EC2 instance type**, choose **t3.nano**.
  - For **Capacity**, enter the minimum number and the maximum number of instances to launch in the Auto Scaling group.

5. (Optional) To manage the cluster tags, expand **Tags**, and then perform one of the following operations:

   [Add a tag] Choose **Add tag** and do the following:

   - For **Key**, enter the key name.
   - For **Value**, enter the key value.

   [Remove a tag] Choose **Remove** to the right of the tag's Key and Value.

6. Choose **Create**.

Step 2: Register a task definition

Before you can run a task on your cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses an `amazonlinux` image from Docker Hub and simply sleeps. For more information about the available task definition parameters, see Amazon ECS task definitions.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the **JSON editor** box, paste the following contents.

```
{
    "family": "ConsoleTutorial-taskdef",
    "containerDefinitions": [
        {
            "name": "sleep",
            "image": "public.ecr.aws/amazonlinux/amazonlinux:latest",
```

```
                "memory": 20,
                "essential": true,
                "command": [
                    "sh",
                    "-c",
                    "sleep infinity"
                ]
            }
        ],
        "requiresCompatibilities": [
            "EC2"
        ]
    }
```

5. Choose **Create**.

Step 3: Run a task

After you have registered a task definition for your account, you can run a task in the cluster. For this tutorial, you run five instances of the `ConsoleTutorial-taskdef` task definition in your `ConsoleTutorial-cluster` cluster.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose **ConsoleTutorial-cluster**.
3. Under **Tasks**, choose **Run new task**.
4. In the **Environment** section, under **Compute options**, choose **Capacity provider strategy**.
5. Under **Deployment configuration**, for **Application type**, choose **Task**.
6. Choose **ConsoleTutorial-taskdef** from the **Family** dropdown list.
7. Under **Desired tasks**, enter 5.
8. Choose **Create**.

Step 4: Verify

At this point in the tutorial, you should have a cluster with five tasks running and an Auto Scaling group with a capacity provider. The capacity provider has Amazon ECS managed scaling enabled.

We can verify that everything is working properly by viewing the CloudWatch metrics, the Auto Scaling group settings, and finally the Amazon ECS cluster task count.

To view the CloudWatch metrics for your cluster

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation bar at the top of the screen, select the Region.
3. On the navigation pane, under **Metrics**, choose **All metrics**.
4. On the **All metrics** page, under the **Browse** tab, choose `AWS/ECS/ManagedScaling`.
5. Choose **CapacityProviderName, ClusterName**.
6. Select the check box that corresponds to the `ConsoleTutorial-cluster` **ClusterName**.
7. Under the **Graphed metrics** tab, change **Period** to **30 seconds** and **Statistic** to **Maximum**.

   The value displayed in the graph shows the target capacity value for the capacity provider. It should begin at `100`, which was the target capacity percent we set. You should see it scale up to `200`, which will trigger an alarm for the target tracking scaling policy. The alarm will then trigger the Auto Scaling group to scale out.

Use the following steps to view your Auto Scaling group details to confirm that the scale-out action occurred.

To verify the Auto Scaling group scaled out

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation bar at the top of the screen, select the Region.
3. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
4. Choose the `ConsoleTutorial-cluster` Auto Scaling group created in this tutorial. View the value under **Desired capacity** and view the instances under the **Instance management** tab to confirm your group scaled out to two instances.

Use the following steps to view your Amazon ECS cluster to confirm that the Amazon EC2 instances were registered with the cluster and your tasks transitioned to a `RUNNING` status.

To verify the instances in the Auto Scaling group

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the `ConsoleTutorial-cluster` cluster.

4. On the **Tasks** tab, confirm you see five tasks in the `RUNNING` status.

Step 5: Clean up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using. Deleting capacity providers and task definitions are not supported, but there is no cost associated with these resources.

To clean up the tutorial resources

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose **ConsoleTutorial-cluster**.
4. On the **ConsoleTutorial-cluster** page, choose the **Tasks** tab, and then choose **Stop**, **Stop all**.
5. In the navigation pane, choose **Clusters**.
6. On the **Clusters** page, choose **ConsoleTutorial-cluster**.
7. In the upper-right of the page, choose **Delete cluster**.
8. In the confirmation box, enter **delete ConsoleTutorial-cluster** and choose **Delete**.
9. Delete the Auto Scaling groups using the following steps.

    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

    b. On the navigation bar at the top of the screen, select the Region.

    c. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.

    d. Select the `ConsoleTutorial-cluster` Auto Scaling group, then choose **Actions**.

    e. From the **Actions** menu, choose **Delete**. Enter **delete** in the confirmation box and then choose **Delete**.

Amazon EC2 container instances for Amazon ECS

An Amazon ECS container instance is an Amazon EC2 instance that run the Amazon ECS container agent and is registered to a cluster. When you run tasks with Amazon ECS using the EC2 launch type, External launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances. You are responsible for the container instance management and maintenance.

Although you can create your own Amazon EC2 instance AMI that meets the basic specifications needed to run your containerized workloads on Amazon ECS, the Amazon ECS-optimized AMIs are preconfigured and tested on Amazon ECS by AWS engineers. It is the simplest way for you to get started and to get your containers running on AWS quickly.

When you create a cluster using the console, Amazon ECS creates a launch template for your instances with the latest AMI associated with the selected operating system.

When you use AWS CloudFormation to create a cluster, the SSM parameter is part of the Amazon EC2 launch template for the Auto Scaling group instances. You can configure the template to use a dynamic Systems Manager parameter to determine what Amazon ECS Optimized AMI to deploy. This parameter ensures that each time you deploy the stack it will check to see if there is available update that needs to be applied to the EC2 instances. For an example of how to use the Systems Manager parameter, see Create an Amazon ECS cluster with the Amazon ECS-optimized Amazon Linux 2023 AMI in the *AWS CloudFormation User Guide*.

- Retrieving Amazon ECS-optimized Linux AMI metadata
- Retrieving Amazon ECS-optimized Bottlerocket AMI metadata
- Retrieving Amazon ECS-optimized Windows AMI metadata

You can choose from the instance types that are compatible with your application. With larger instances, you can launch more tasks at the same time. With smaller instances, you can scale out in a more fine-grained way to save costs. You don't need to choose a single Amazon EC2 instance type that to fit all the applications in your cluster. Instead, you can create multiple Auto Scaling groups where each group has a different instance type. Then, you can create an Amazon EC2 capacity provider for each one of these groups.

Use the following guidelines to determine the instance family types and instance type to use:

- Eliminate the instance types or instance families that don't meet the specific requirements of your application. For example, if your application requires a GPU, you can exclude any instance types that don't have a GPU.
- Consider requirements including network throughput and storage.
- Consider the CPU and memory. As a general rule, the CPU and memory must be large enough to hold at least one replica of the task that you want to run.

Spot Instances

Spot capacity can provide significant cost savings over on-demand instances. Spot capacity is excess capacity that's priced significantly lower than on-demand or reserved capacity. Spot capacity is suitable for batch processing and machine-learning workloads, and development and staging environments. More generally, it's suitable for any workload that tolerates temporary downtime.

Understand that the following consequences because Spot capacity might not be available all the time.

- During periods of extremely high demand, Spot capacity might be unavailable. This can cause Amazon EC2 Spot instance launches to be delayed. In these events, Amazon ECS services retry launching tasks, and Amazon EC2 Auto Scaling groups also retry launching instances, until the required capacity becomes available. Amazon EC2 doesn't replace Spot capacity with on-demand capacity.
- When the overall demand for capacity increases, Spot Instances and tasks might be terminated with only a two-minute warning. After the warning is sent, tasks should begin an orderly shutdown if necessary before the instance is fully terminated. This helps minimize the possibility of errors. For more information about a graceful shutdown, see Graceful shutdowns with ECS.

To help minimize Spot capacity shortages, consider the following recommendations:

- Use multiple Regions and Availability Zones - Spot capacity varies by Region and Availability Zone. You can improve Spot availability by running your workloads in multiple Regions and Availability Zones. If possible, specify subnets in all the Availability Zones in the Regions where you run your tasks and instances.
- Use multiple Amazon EC2 instance types - When you use Mixed Instance Policies with Amazon EC2 Auto Scaling, multiple instance types are launched into your Auto Scaling Group. This ensures that a request for Spot capacity can be fulfilled when needed. To maximize reliability and minimize complexity, use instance types with roughly the same amount of CPU and memory in your Mixed Instances Policy. These instances can be from a different generation, or variants of the same base instance type. Note that they might come with additional features that you might not require. An example of such a list could include m4.large, m5.large, m5a.large, m5d.large, m5n.large, m5dn.large, and m5ad.large. For more information, see Auto Scaling groups with multiple instance types and purchase options in the *Amazon EC2 Auto Scaling User Guide*.
- Use the capacity-optimized Spot allocation strategy - With Amazon EC2 Spot, you can choose between the capacity- and cost-optimized allocation strategies. If you choose the capacity-

optimized strategy when launching a new instance, Amazon EC2 Spot selects the instance type with the greatest availability in the selected Availability Zone. This helps reduce the possibility that the instance is terminated soon after it launches.

For information about how to configure spot termination notices on your container instances, see:

- Configuring Amazon ECS Linux container instances to receive Spot Instance notices
- Configuring Amazon ECS Windows container instances to receive Spot Instance notices

Amazon ECS-optimized Linux AMIs

Important

The Amazon ECS-Optimized Amazon Linux 2 AMI reaches end-of-life on June 30, 2026, mirroring the same EOL date of the upstream Amazon Linux 2 operating system (for more information, see the Amazon Linux 2 FAQs). We encourage customers to upgrade their applications to use Amazon Linux 2023, which includes long term support through 2028. For information about migrating from Amazon Linux 2 to Amazon Linux 2023, see Migrating from the Amazon Linux 2 Amazon ECS-optimized AMI to the Amazon Linux 2023 Amazon ECS-optimized AMI.

Amazon ECS provides the Amazon ECS-optimized AMIs that are preconfigured with the requirements and recommendations to run your container workloads. We recommend that you use the Amazon ECS-optimized Amazon Linux 2023 AMI for your Amazon EC2 instances. Launching your container instances from the most recent Amazon ECS-Optimized AMI ensures that you receive the current security updates and container agent version. For information about how to launch an instance, see Launching an Amazon ECS Linux container instance.

When you create a cluster using the console, Amazon ECS creates a launch template for your instances with the latest AMI associated with the selected operating system.

When you use AWS CloudFormation to create a cluster, the SSM parameter is part of the Amazon EC2 launch template for the Auto Scaling group instances. You can configure the template to use a dynamic Systems Manager parameter to determine what Amazon ECS Optimized AMI to deploy. This parameter ensures that each time you deploy the stack it will check to see if there is available update that needs to be applied to the EC2 instances. For an example of how to use the Systems

Manager parameter, see Create an Amazon ECS cluster with the Amazon ECS-optimized Amazon Linux 2023 AMI in the *AWS CloudFormation User Guide*.

If you need to customize the Amazon ECS-optimized AMI, see Amazon ECS Optimized AMI Build Recipes on GitHub.

The following variants of the Amazon ECS-optimized AMI are available for your Amazon EC2 instances with the Amazon Linux 2023 operating system.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2023	**Amazon ECS-optimized Amazon Linux 2023 AMI**	Amazon Linux 2023 is the next generation of Amazon Linux from AWS. For most cases, recommended for launching your Amazon EC2 instances for your Amazon ECS workloads. For more information, see What is Amazon Linux 2023 in the *Amazon Linux 2023 User Guide*.	By default, the Amazon ECS-optimized Amazon Linux 2023 AMI ships with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2023 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more

Operating system	AMI	Description	Storage configuration
			information, see Use the OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2023 (arm64)	**Amazon ECS-optimized Amazon Linux 2023 (arm64) AMI**	Based on Amazon Linux 2023, this AMI is recommended for use when launching your Amazon EC2 instances, which are powered by Arm-based AWS Graviton/Graviton 2/Graviton 3/Graviton 4 Processors, for your Amazon ECS workloads. For more information, see [Specifications for the Amazon EC2 general purpose instances]() in the *Amazon EC2 Instance Types guide*.	By default, the Amazon ECS-optimized Amazon Linux 2023 AMI ships with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2023 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the OverlayFS storage driver]() in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2023 (Neuron)	**Amazon ECS-optimized Amazon Linux 2023 AMI**	Based on Amazon Linux 2023, this AMIis for Amazon EC2 Inf1, Trn1 or Inf2 instances. It comes pre-configured with AWS Inferentia and AWS Trainium drivers and the AWS Neuron runtime for Docker which makes running machine learning inference workloads easier on Amazon ECS. For more information, see Amazon ECS task definitions for AWS Neuron machine learning workloads. The Amazon ECS-optimized Amazon Linux 2023 (Neuron) AMI does not come with the AWS CLI preinstalled.	By default, the Amazon ECS-optimized Amazon Linux 2023 AMI ships with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2023 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see Use the OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2023 GPU	**Amazon ECS optimized Amazon Linux 2023 GPU AMI**	Based on Amazon Linux 2023, this AMI is recommended for use when launching your Amazon EC2 GPU-based instances for your Amazon ECS workloads. It comes pre-configured with NVIDIA kernel drivers and a Docker GPU runtime which makes running workloads that take advantage of GPUs on Amazon ECS. For more information, see [Amazon ECS task definitions for GPU workloads]().	By default, the Amazon ECS-optimized Amazon Linux 2023 AMI ships with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2023 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the OverlayFS storage driver]() in the Docker documentation.

The following variants of the Amazon ECS-optimized AMI are available for your Amazon EC2 instances with the Amazon Linux 2 operating system.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2	**Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMI**	Based on Amazon Linux 2, this AMI is for use when launching your Amazon EC2 instances and you want to use Linux kernel 5.10 instead of kernel 4.14 for your Amazon ECS workloads. The Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMI does not come with the AWS CLI preinstalled.	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2`

Operating system	AMI	Description	Storage configuration
			storage driver. For more information, see Use the OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2	**Amazon ECS-optimized Amazon Linux 2 AMI**	This is for your Amazon ECS workloads. The Amazon ECS-optimized Amazon Linux 2 AMI does not come with the AWS CLI preinstalled.	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see Use the

Operating system	AMI	Description	Storage configuration
			OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2 (arm64)	**Amazon ECS-optimized Amazon Linux 2 kernel 5.10 (arm64) AMI**	Based on Amazon Linux 2, this AMI is for your Amazon EC2 instances, which are powered by Arm-based AWS Graviton/Graviton 2/Graviton 3/Graviton 4 Processors, and you want to use Linux kernel 5.10 instead of Linux kernel 4.14 for your Amazon ECS workloads. For more information, see [Specifications for Amazon EC2 general purpose instances](#) in the *Amazon EC2 Instance Types guide*. The Amazon ECS-optimized Amazon Linux 2 (arm64) AMI does not come with the AWS CLI preinstalled.	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the](#)

Operating system	AMI	Description	Storage configuration
			OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2 (arm64)	**Amazon ECS-optimized Amazon Linux 2 (arm64) AMI**	Based on Amazon Linux 2, this AMI is for use when launching your Amazon EC2 instances, which are powered by Arm-based AWS Graviton/Graviton 2/Graviton 3/Graviton 4 Processors, for your Amazon ECS workloads. The Amazon ECS-optimized Amazon Linux 2 (arm64) AMI does not come with the AWS CLI preinstalled.	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see Use the

Operating system	AMI	Description	Storage configuration
			OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2 (GPU)	**Amazon ECS GPU-optimized kernel 5.10 AMI**	Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 GPU-based instances with Linux kernel 5.10 for your Amazon ECS workloads. It comes pre-configured with NVIDIA kernel drivers and a Docker GPU runtime which makes running workloads that take advantage of GPUs on Amazon ECS. For more information, see [Amazon ECS task definitions for GPU workloads]().	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the]()

Operating system	AMI	Description	Storage configuration
			OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2 (GPU)	**Amazon ECS GPU-optimized AMI**	Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 GPU-based instances with Linux kernel 4.14 for your Amazon ECS workloads. It comes pre-configured with NVIDIA kernel drivers and a Docker GPU runtime which makes running workloads that take advantage of GPUs on Amazon ECS. For more information, see [Amazon ECS task definitions for GPU workloads]().	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the]()

Operating system	AMI	Description	Storage configuration
			OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2 (Neuron)	**Amazon ECS optimized Amazon Linux 2 (Neuron) kernel 5.10 AMI**	Based on Amazon Linux 2, this AMI is for Amazon EC2 Inf1, Trn1 or Inf2 instances. It comes pre-configured with AWS Inferentia with Linux kernel 5.10 and AWS Trainium drivers and the AWS Neuron runtime for Docker which makes running machine learning inference workloads easier on Amazon ECS. For more information, see [Amazon ECS task definitions for AWS Neuron machine learning workloads](). The Amazon ECS optimized Amazon Linux 2 (Neuron) AMI does not come with the AWS CLI preinstalled.	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the]()

Operating system	AMI	Description	Storage configuration
			OverlayFS storage driver in the Docker documentation.

Operating system	AMI	Description	Storage configuration
Amazon Linux 2 (Neuron)	**Amazon ECS optimized Amazon Linux 2 (Neuron) AMI**	Based on Amazon Linux 2, this AMI is for Amazon EC2 Inf1, Trn1 or Inf2 instances. It comes pre-configured with AWS Inferentia and AWS Trainium drivers and the AWS Neuron runtime for Docker which makes running machine learning inference workloads easier on Amazon ECS. For more information, see [Amazon ECS task definitions for AWS Neuron machine learning workloads](). The Amazon ECS optimized Amazon Linux 2 (Neuron) AMI does not come with the AWS CLI preinstalled.	By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `xfs`, and Docker uses the `overlay2` storage driver. For more information, see [Use the]()

Operating system	AMI	Description	Storage configuration
			OverlayFS storage driver in the Docker documentation.

Amazon ECS provides a changelog for the Linux variant of the Amazon ECS-optimized AMI on GitHub. For more information, see Changelog.

The Linux variants of the Amazon ECS-optimized AMI use the Amazon Linux 2 AMI or Amazon Linux 2023 AMI as their base. You can retrieve the AMI name for each variant by querying the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-optimized Linux AMI metadata. The Amazon Linux 2 AMI release notes are available as well. For more information, see Amazon Linux 2 release notes. The Amazon Linux 2023 release notes are available as well. For more information see, Amazon Linux 2023 release notes.

The following pages provide additional information about the changes:

- Source AMI release notes on GitHub
- Docker Engine release notes in the Docker documentation
- NVIDIA Driver Documentation in the NVIDIA documentation
- Amazon ECS agent changelog on GitHub

  The source code for the `ecs-init` application and the scripts and configuration for packaging the agent are now part of the agent repository. For older versions of `ecs-init` and packaging, see Amazon ecs-init changelog on GitHub

Applying security updates to the Amazon ECS-optimized AMI

The Amazon ECS-optimized AMIs based on Amazon Linux contain a customized version of cloud-init. Cloud-init is a package that is used to bootstrap Linux images in a cloud computing environment and perform desired actions when launching an instance. By default, all Amazon ECS-optimized AMIs based on Amazon Linux released before June 12, 2024 have all "Critical" and "Important" security updates applied upon instance launch.

Beginning with the June 12, 2024 releases of the Amazon ECS-optimized AMIs based on Amazon Linux 2, the default behavior will no longer include updating packages at launch. Instead, we recommend that you update to a new Amazon ECS-optimized AMI as releases are made available. The Amazon ECS-optimized AMIs are released when there are available security updates or base AMI changes. This will ensure you are receiving the latest package versions and security updates, and that the package versions are immutable through instance launches. For more information on retrieving the latest Amazon ECS-optimized AMI, see Retrieving Amazon ECS-optimized Linux AMI metadata.

We recommend automating your environment to update to a new AMI as they are made available. For information about the available options, see Amazon ECS enables easier EC2 capacity management, with managed instance draining.

To continue applying "Critical" and "Important" security updates manually on an AMI version, you can run the following command on your Amazon EC2 instance.

```
yum update --security
```

If you want to re-enable security updates at launch, you can add the following line to the `#cloud-config` section of the cloud-init user data when launching your Amazon EC2 instance. For more information, see Using cloud-init on Amazon Linux 2 in the *Amazon Linux User Guide*.

```
#cloud-config
repo_upgrade: security
```

Retrieving Amazon ECS-optimized Linux AMI metadata

You can programmatically retrieve the Amazon ECS-optimized AMI metadata. The metadata includes the AMI name, Amazon ECS container agent version, and Amazon ECS runtime version which includes the Docker version.

When you create a cluster using the console, Amazon ECS creates a launch template for your instances with the latest AMI associated with the selected operating system.

When you use AWS CloudFormation to create a cluster, the SSM parameter is part of the Amazon EC2 launch template for the Auto Scaling group instances. You can configure the template to use a dynamic Systems Manager parameter to determine what Amazon ECS Optimized AMI to deploy. This parameter ensures that each time you deploy the stack it will check to see if there is available update that needs to be applied to the EC2 instances. For an example of how to use the Systems Manager parameter, see Create an Amazon ECS cluster with the Amazon ECS-optimized Amazon Linux 2023 AMI in the *AWS CloudFormation User Guide*.

The AMI ID, image name, operating system, container agent version, source image name, and runtime version for each variant of the Amazon ECS-optimized AMIs can be programmatically retrieved by querying the Systems Manager Parameter Store API. For more information about the Systems Manager Parameter Store API, see GetParameters and GetParametersByPath.

> **Note**
>
> Your administrative user must have the following IAM permissions to retrieve the Amazon ECS-optimized AMI metadata. These permissions have been added to the `AmazonECS_FullAccess` IAM policy.
>
> - ssm:GetParameters
> - ssm:GetParameter
> - ssm:GetParametersByPath

Systems Manager Parameter Store parameter format

The following is the format of the parameter name for each Amazon ECS-optimized AMI variant.

Linux Amazon ECS-optimized AMIs

- Amazon Linux 2023 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/<version>
```

- Amazon Linux 2023 (arm64) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/arm64/<version>
```

- Amazon Linux 2023 (Neuron) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/neuron/<version>
```

- Amazon Linux 2023 (GPU) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2023/gpu/<version>
```

  Amazon Linux 2 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/<version>
```

- Amazon Linux 2 kernel 5.10 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/<version>
```

- Amazon Linux 2 (arm64) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/<version>
```

- Amazon Linux 2 kernel 5.10 (arm64) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/arm64/<version>
```

- Amazon ECS GPU-optimized kernel 5.10 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/gpu/<version>
```

- Amazon Linux 2 (GPU) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/<version>
```

- Amazon ECS optimized Amazon Linux 2 (Neuron) kernel 5.10 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/inf/<version>
```

- Amazon Linux 2 (Neuron) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/inf/<version>
```

The following parameter name format retrieves the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2 AMI by using the sub-parameter `image_id`.

```
/aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id
```

The following parameter name format retrieves the metadata of a specific Amazon ECS-optimized AMI version by specifying the AMI name.

- Amazon ECS-optimized Amazon Linux 2 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-ecs-hvm-2.0.20181112-x86_64-
ebs
```

> **Note**
>
> All versions of the Amazon ECS-optimized Amazon Linux 2 AMI are available for retrieval. Only Amazon ECS-optimized AMI versions `amzn-ami-2017.09.l-amazon-ecs-optimized` (Linux) and later can be retrieved.

Examples

The following examples show ways in which you can retrieve the metadata for each Amazon ECS-optimized AMI variant.

Retrieving the metadata of the latest recommended Amazon ECS-optimized AMI

You can retrieve the latest recommended Amazon ECS-optimized AMI using the AWS CLI with the following AWS CLI commands.

Linux Amazon ECS-optimized AMIs

- **For the Amazon ECS-optimized Amazon Linux 2023 AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2023 (arm64) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
arm64/recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2023 (Neuron) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
neuron/recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2023 GPU AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/gpu/
recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2 AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2 kernel 5.10 (arm64) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/arm64/recommended --region us-east-1
```

- **For the Amazon ECS-optimized Amazon Linux 2 (arm64) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/arm64/
recommended --region us-east-1
```

- **For the Amazon ECS GPU-optimized kernel 5.10 AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/gpu/recommended --region us-east-1
```

- **For the Amazon ECS GPU-optimized AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/
recommended --region us-east-1
```

- **For the Amazon ECS optimized Amazon Linux 2 (Neuron) kernel 5.10 AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/inf/recommended --region us-east-1
```

- **For the Amazon ECS optimized Amazon Linux 2 (Neuron) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/inf/
recommended --region us-east-1
```

Retrieving the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2023 AMI

You can retrieve the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2023 AMI ID by using the sub-parameter `image_id`.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-
linux-2023/recommended/image_id --region us-east-1
```

To retrieve the `image_id` value only, you can query the specific parameter value; for example:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2023/
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

Retrieving the metadata of a specific Amazon ECS-optimized Amazon Linux 2 AMI version

Retrieve the metadata of a specific Amazon ECS-optimized Amazon Linux AMI version using the AWS CLI with the following AWS CLI command. Replace the AMI name with the name of the Amazon ECS-optimized Amazon Linux AMI to retrieve.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-
ecs-hvm-2.0.20200928-x86_64-ebs --region us-east-1
```

Retrieving the Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMI metadata using the Systems Manager GetParametersByPath API

Retrieve the Amazon ECS-optimized Amazon Linux 2 AMI metadata with the Systems Manager GetParametersByPath API using the AWS CLI with the following command.

```
aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/ --region us-east-1
```

Retrieving the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMI

You can retrieve the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2 kernel 5.10 AMI ID by using the sub-parameter `image_id`.

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
kernel-5.10/recommended/image_id --region us-east-1
```

To retrieve the `image_id` value only, you can query the specific parameter value; for example:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template

You can reference the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template by referencing the Systems Manager parameter store name.

Linux example

```
Parameters:kernel-5.10
  LatestECSOptimizedAMI:
    Description: AMI ID
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ecs/optimized-ami/amazon-linux-2/kernel-5.10/recommended/
image_id
```

Migrating from an Amazon Linux 2 to an Amazon Linux 2023 Amazon ECS-optimized AMI

Following Amazon Linux, Amazon ECS ends standard support for Amazon Linux 2 Amazon ECS-optimized AMIs effective June 30, 2026. After this date, the Amazon ECS agent version is pinned and new Amazon Linux 2 Amazon ECS-optimized AMIs are only published when the source Amazon Linux 2 AMI is updated. Complete End of Life (EOL) occurs on June 30, 2026, after which no more Amazon ECS-optimized Amazon Linux 2 AMIs are published, even if the source AMI is updated.

Amazon Linux 2023 provides a secure-by-default approach with preconfigured security policies, SELinux in permissive mode, IMDSv2-only mode enabled by default, optimized boot times, and improved package management for enhanced security and performance.

There is a high degree of compatibility between the Amazon Linux 2 and Amazon Linux 2023 Amazon ECS-optimized AMIs, and most customers will experience minimal-to-zero changes in their workloads between the two operating systems.

For more information, see [Comparing Amazon Linux 2 and *Amazon Linux 2023*] in the *Amazon Linux 2023 User Guide* and the [AL2023 FAQs].

Compatibility considerations

Package management and OS updates

Unlike previous versions of Amazon Linux, Amazon ECS-optimized Amazon Linux 2023 AMIs are locked to a specific version of the Amazon Linux repository. This insulates users from inadvertently updating packages that might bring in unwanted or breaking changes. For more information, see [Managing repositories and OS updates in Amazon Linux 2023] in the *Amazon Linux 2023 User Guide*.

Linux kernel versions

Amazon Linux 2 AMIs are based on Linux kernels 4.14 and 5.10, while Amazon Linux 2023 uses Linux kernel 6.1 and 6.12. For more information, see [Comparing Amazon Linux 2 and Amazon Linux 2023 kernels] in the *Amazon Linux 2023 User Guide*.

Package availability changes

The following are notable package changes in Amazon Linux 2023:

- Some source binary packages in Amazon Linux 2 are no longer available in Amazon Linux 2023. For more information, see [Packages removed from Amazon Linux 2023] in the *Amazon Linux 2023 Release Notes*.
- Changes in how Amazon Linux supports different versions of packages. The `amazon-linux-extras` system used in Amazon Linux 2 does not exist in Amazon Linux 2023. All packages are simply available in the "core" repository.
- Extra packages for Enterprise Linux (EPEL) are not supported in Amazon Linux 2023. For more information, see [EPEL compatibility in Amazon Linux 2023] in the *Amazon Linux 2023 User Guide*.
- 32-bit applications are not supported in Amazon Linux 2023. For more information, see [Deprecated features from Amazon Linux 2] in the *Amazon Linux 2023 User Guide*.

Control Groups (cgroups) changes

A Control Group (cgroup) is a Linux kernel feature to hierarchically organize processes and distribute system resources between them. Control Groups are used extensively to implement a container runtime, and by `systemd`.

The Amazon ECS agent, Docker, and containerd all support both cgroupv1 and cgroupv2. Amazon ECS agent and the container runtime manage cgroups for you, so Amazon ECS customers do not need to make any changes for this underlying cgroup upgrade.

For further details on cgroupv2, see [Control groups v2 in Amazon Linux 2023]() in the *Amazon Linux 2023 User Guide*.

Instance Metadata Service (IMDS) changes

Amazon Linux 2023 requires Instance Metadata Service version 2 (IMDSv2) by default. IMDSv2 has several benefits that help improve security posture. It uses a session-oriented authentication method that requires the creation of a secret token in a simple HTTP PUT request to start the session. A session's token can be valid for anywhere between 1 second and 6 hours.

For more information on how to transition from IMDSv1 to IMDSv2, see [Transition to using Instance Metadata Service Version 2]() in the *Amazon EC2 User Guide*.

If you would like to use IMDSv1, you can still do so by manually overriding the settings using instance metadata option launch properties.

Memory swappiness changes

Per-container memory swappiness is not supported on Amazon Linux 2023 and cgroups v2. For more information, see [Managing container swap memory space on Amazon ECS]().

FIPS validation changes

Amazon Linux 2 is certified under FIPS 140-2 and Amazon Linux 2023 is certified under FIPS 140-3.

To enable FIPS mode on Amazon Linux 2023, install the necessary packages on your Amazon EC2 instance and follow the configuration steps using the instructions in [Enable FIPS Mode on Amazon Linux 2023]() in the *Amazon Linux 2023 User Guide*.

Accelerated instance support

The Amazon ECS-optimized Amazon Linux 2023 AMIs support both Neuron and GPU accelerated instance types. For more information, see [Amazon ECS-optimized Linux AMIs]().

Building custom AMIs

While we recommend moving to officially supported and published Amazon ECS-optimized AMIs for Amazon Linux 2023, you can continue to build custom Amazon Linux 2 Amazon ECS-optimized AMIs using the open-source build scripts that are used to build the Linux variants of the Amazon ECS-optimized AMI. For more information, see Amazon ECS-optimized Linux AMI build script.

Migration strategies

We recommend creating and implementing a migration plan that includes thorough application testing. The following sections outline different migration strategies based on how you manage your Amazon ECS infrastructure.

Migrating with Amazon ECS capacity providers

1. Create a new capacity provider with a new launch template. This should reference an Auto Scaling group with a launch template similar to your existing one, but instead of the Amazon Linux 2 Amazon ECS-optimized AMI, it should specify one of the Amazon Linux 2023 variants. Add this new capacity provider to your existing Amazon ECS cluster.
2. Update your cluster's default capacity provider strategy to include both the existing Amazon Linux 2 capacity provider and the new Amazon Linux 2023 capacity provider. Start with a higher weight on the Amazon Linux 2 provider and a lower weight on the Amazon Linux 2023 provider (for example, Amazon Linux 2: weight 80, Amazon Linux 2023: weight 20). This causes Amazon ECS to begin provisioning Amazon Linux 2023 instances as new tasks are scheduled. Verify that the instances register correctly and that tasks are able to run successfully on the new instances.
3. Gradually adjust the capacity provider weights in your cluster's default strategy, increasing the weight for the Amazon Linux 2023 provider while decreasing the Amazon Linux 2 provider weight over time (for example, 60/40, then 40/60, then 20/80). You can also update individual service capacity provider strategies to prioritize Amazon Linux 2023 instances. Monitor task placement to ensure they're successfully running on Amazon Linux 2023 instances.
4. Optionally drain Amazon Linux 2 container instances to accelerate task migration. If you have sufficient Amazon Linux 2023 replacement capacity, you can manually drain your Amazon Linux 2 container instances through the Amazon ECS console or AWS CLI to speed up the transition of your tasks from Amazon Linux 2 to Amazon Linux 2023. After the migration is complete, remove the Amazon Linux 2 capacity provider from your cluster and delete the associated Auto Scaling group.

Migrating with an Amazon EC2 Auto Scaling group

1. Create a new Amazon EC2 Auto Scaling group with a new launch template. This should be similar to your existing launch template, but instead of the Amazon Linux 2 Amazon ECS-optimized AMI, it should specify one of the Amazon Linux 2023 variants. This new Auto Scaling group can launch instances to your existing cluster.
2. Scale up the Auto Scaling group so that you begin to have Amazon Linux 2023 instances registering to your cluster. Verify that the instances register correctly and that tasks are able to run successfully on the new instances.
3. After your tasks have been verified to work on Amazon Linux 2023, scale up the Amazon Linux 2023 Auto Scaling group while gradually scaling down the Amazon Linux 2 Auto Scaling group, until you have completely replaced all Amazon Linux 2 instances.
4. If you have sufficient Amazon Linux 2023 replacement capacity, you might want to explicitly drain the container instances to speed up the transition of your tasks from Amazon Linux 2 to Amazon Linux 2023. For more information, see Draining Amazon ECS container instances.

Migrating with manually managed instances

1. Manually launch (or adjust scripts that launch) new Amazon EC2 instances using the Amazon ECS-optimized Amazon Linux 2023 AMI instead of Amazon Linux 2. Ensure these instances use the same security groups, subnets, IAM roles, and cluster configuration as your existing Amazon Linux 2 instances. The instances should automatically register to your existing Amazon ECS cluster upon launch.
2. Verify the new Amazon Linux 2023 instances are successfully registering to your Amazon ECS cluster and are in an `ACTIVE` state. Test that tasks can be scheduled and run properly on these new instances by either waiting for natural task placement or manually stopping/starting some tasks to trigger rescheduling.
3. Gradually replace your Amazon Linux 2 instances by launching additional Amazon Linux 2023 instances as needed, then manually draining and terminating the Amazon Linux 2 instances one by one. You can drain instances through the Amazon ECS console by setting the instance to `DRAINING` status, which will stop placing new tasks on it and allow existing tasks to finish or be rescheduled elsewhere.

Amazon ECS-optimized Linux AMI build script

Amazon ECS has open-sourced the build scripts that are used to build the Linux variants of the Amazon ECS-optimized AMI. These build scripts are now available on GitHub. For more information, see amazon-ecs-ami on GitHub.

If you need to customize the Amazon ECS-optimized AMI , see Amazon ECS Optimized AMI Build Recipies on GitHub.

The build scripts repository includes a HashiCorp packer template and build scripts to generate each of the Linux variants of the Amazon ECS-optimized AMI. These scripts are the source of truth for Amazon ECS-optimized AMI builds, so you can follow the GitHub repository to monitor changes to our AMIs. For example, perhaps you want your own AMI to use the same version of Docker that the Amazon ECS team uses for the official AMI.

For more information, see the Amazon ECS AMI repository at aws/amazon-ecs-ami on GitHub.

To build an Amazon ECS-optimized Linux AMI

1. Clone the `aws/amazon-ecs-ami` GitHub repo.

   ```
   git clone https://github.com/aws/amazon-ecs-ami.git
   ```

2. Add an environment variable for the AWS Region to use when creating the AMI. Replace the `us-west-2` value with the Region to use.

   ```
   export REGION=us-west-2
   ```

3. A Makefile is provided to build the AMI. From the root directory of the cloned repository, use one of the following commands, corresponding to the Linux variant of the Amazon ECS-optimized AMI you want to build.

   - Amazon ECS-optimized Amazon Linux 2 AMI

     ```
     make al2
     ```

   - Amazon ECS-optimized Amazon Linux 2 (arm64) AMI

     ```
     make al2arm
     ```

   - Amazon ECS GPU-optimized AMI

```
make al2gpu
```

- Amazon ECS optimized Amazon Linux 2 (Neuron) AMI

```
make al2inf
```

- Amazon ECS-optimized Amazon Linux 2023 AMI

```
make al2023
```

- Amazon ECS-optimized Amazon Linux 2023 (arm64) AMI

```
make al2023arm
```

- Amazon ECS optimized Amazon Linux 2023 (Neuron) AMI

```
make al2023neu
```

Amazon ECS-optimized Bottlerocket AMIs

Bottlerocket is a Linux based open-source operating system that is purpose built by AWS for running containers on virtual machines or bare metal hosts. The Amazon ECS-optimized Bottlerocket AMI is secure and only includes the minimum number of packages that's required to run containers. This improves resource usage, reduces security attack surface, and helps lower management overhead. The Bottlerocket AMI is also integrated with Amazon ECS to help reduce the operational overhead involved in updating container instances in a cluster.

Bottlerocket differs from Amazon Linux in the following ways:

- Bottlerocket doesn't include a package manager, and its software can only be run as containers. Updates to Bottlerocket are both applied and can be rolled back in a single step, which reduces the likelihood of update errors.
- The primary mechanism to manage Bottlerocket hosts is with a container scheduler. Unlike Amazon Linux, logging into individual Bottlerocket instances is intended to be an infrequent operation for advanced debugging and troubleshooting purposes only.

For more information about Bottlerocket, see the documentation and releases on GitHub.

There are variants of the Amazon ECS-optimized Bottlerocket AMI for kernel 6.1 and kernel 5.10.

The following variants use kernel 6.1:

- `aws-ecs-2`
- `aws-ecs-2-nvidia`

The following variants use kernel 5.10:

- `aws-ecs-1`
- `aws-ecs-1-nvidia`

  For more information about the `aws-ecs-1-nvidia` variant, see Announcing NVIDIA GPU support for Bottlerocket on Amazon ECS.

Considerations

Consider the following when using a Bottlerocket AMI with Amazon ECS.

- Bottlerocket supports Amazon EC2 instances with `x86_64` and `arm64` processors. The Bottlerocket AMI isn't recommended for use with Amazon EC2 instances with an Inferentia chip.
- Bottlerocket images don't include an SSH server or a shell. However, you can use out-of-band management tools to gain SSH administrator access and perform bootstrapping. For more information, see these sections in the bottlerocket README.md on GitHub:
  - Exploration
  - Admin container
- By default, Bottlerocket has a control container that's enabled. This container runs the AWS Systems Manager agent that you can use to run commands or start shell sessions on Amazon EC2 Bottlerocket instances. For more information, see Setting up Session Manager in the *AWS Systems Manager User Guide*.
- Bottlerocket is optimized for container workloads and has a focus on security. Bottlerocket doesn't include a package manager and is immutable. For information about the security features and guidance, see Security Features and Security Guidance on GitHub.
- The `awsvpc` network mode is supported for Bottlerocket AMI version `1.1.0` or later.
- App Mesh in a task definition is supported for Bottlerocket AMI version `1.15.0` or later.

- The `initProcessEnabled` task definition parameter is supported for Bottlerocket AMI version `1.19.0` or later.
- The Bottlerocket AMIs also don't support the following services and features:
  - ECS Anywhere
  - Service Connect
  - Amazon EFS in encrypted mode
  - Amazon EFS in `awsvpc` network mode
  - Amazon EBS volumes can't be mounted
  - Elastic Inference Accelerator

Retrieving Amazon ECS-optimized Bottlerocket AMI metadata

You can retrieve the Amazon Machine Image (AMI) ID for Amazon ECS-optimized AMIs by querying the AWS Systems Manager Parameter Store API. Using this parameter, you don't need to manually look up Amazon ECS-optimized AMI IDs. For more information about the Systems Manager Parameter Store API, see GetParameter. The user that you use must have the `ssm:GetParameter` IAM permission to retrieve the Amazon ECS-optimized AMI metadata.

`aws-ecs-2` Bottlerocket AMI variant

You can retrieve the latest stable `aws-ecs-2` Bottlerocket AMI variant by AWS Region and architecture with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon ECS-optimized Bottlerocket AMI with the following AWS CLI command by using the subparameter `image_id`. Replace the *`region`* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see Finding an AMI on GitHub. To retrieve a version other than the latest, replace `latest` with the version number.
  - For the 64-bit (`x86_64`) architecture:

    ```
    aws ssm get-parameter --region us-east-2 --name "/aws/service/bottlerocket/aws-ecs-2/x86_64/latest/image_id" --query Parameter.Value --output text
    ```

  - For the 64-bit Arm (`arm64`) architecture:

    ```
    aws ssm get-parameter --region us-east-2 --name "/aws/service/bottlerocket/aws-ecs-2/arm64/latest/image_id" --query Parameter.Value --output text
    ```

- **AWS Management Console** – You can query for the recommended Amazon ECS-optimized AMI ID using a URL in the AWS Management Console. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace *region* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see Finding an AMI on GitHub.
  - For the 64-bit (x86_64) architecture:

    ```
    https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
    aws-ecs-2/x86_64/latest/image_id/description?region=region#
    ```

  - For the 64-bit Arm (arm64) architecture:

    ```
    https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
    aws-ecs-2/arm64/latest/image_id/description?region=region#
    ```

aws-ecs-2-nvidia Bottlerocket AMI variant

You can retrieve the latest stable aws-ecs-2-nvdia Bottlerocket AMI variant by Region and architecture with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon ECS-optimized Bottlerocket AMI with the following AWS CLI command by using the subparameter image_id. Replace the *region* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see Finding an AMI on GitHub. To retrieve a version other than the latest, replace latest with the version number.
  - For the 64-bit (x86_64) architecture:

    ```
    aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
    ecs-2-nvidia/x86_64/latest/image_id" --query Parameter.Value --output text
    ```

  - For the 64 bit Arm (arm64) architecture:

    ```
    aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
    ecs-2-nvidia/arm64/latest/image_id" --query Parameter.Value --output text
    ```

- **AWS Management Console** – You can query for the recommended Amazon ECS optimized AMI ID using a URL in the AWS Management Console. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace *region* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub.

- For the 64 bit (`x86_64`) architecture:

```
https://regionconsole.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-2-nvidia/x86_64/latest/image_id/description?region=region#
```

- For the 64 bit Arm (`arm64`) architecture:

```
https://regionconsole.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-2-nvidia/arm64/latest/image_id/description?region=region#
```

`aws-ecs-1` Bottlerocket AMI variant

You can retrieve the latest stable `aws-ecs-1` Bottlerocket AMI variant by AWS Region and architecture with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon ECS-optimized Bottlerocket AMI with the following AWS CLI command by using the subparameter `image_id`. Replace the *region* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub. To retrieve a version other than the latest, replace `latest` with the version number.
  - For the 64-bit (`x86_64`) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1/x86_64/latest/image_id" --query Parameter.Value --output text
```

  - For the 64-bit Arm (`arm64`) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1/arm64/latest/image_id" --query Parameter.Value --output text
```

- **AWS Management Console** – You can query for the recommended Amazon ECS-optimized AMI ID using a URL in the AWS Management Console. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace *region* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub.
  - For the 64-bit (`x86_64`) architecture:

```
https://region.console.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-1/x86_64/latest/image_id/description
```

- For the 64-bit Arm (`arm64`) architecture:

```
https://region.console.aws.amazon.com/systems-manager/parameters/aws/service/
bottlerocket/aws-ecs-1/arm64/latest/image_id/description
```

`aws-ecs-1-nvidia` Bottlerocket AMI variant

You can retrieve the latest stable `aws-ecs-1-nvdia` Bottlerocket AMI variant by Region and architecture with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon ECS-optimized Bottlerocket AMI with the following AWS CLI command by using the subparameter `image_id`. Replace the *`region`* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub. To retrieve a version other than the latest, replace `latest` with the version number.
  - For the 64-bit (`x86_64`) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1-nvidia/x86_64/latest/image_id" --query Parameter.Value --output text
```

  - For the 64 bit Arm (`arm64`) architecture:

```
aws ssm get-parameter --region us-east-1 --name "/aws/service/bottlerocket/aws-
ecs-1-nvidia/arm64/latest/image_id" --query Parameter.Value --output text
```

- **AWS Management Console** – You can query for the recommended Amazon ECS optimized AMI ID using a URL in the AWS Management Console. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace *`region`* with the Region code that you want the AMI ID for. For information about the supported AWS Regions, see [Finding an AMI](#) on GitHub.
  - For the 64 bit (`x86_64`) architecture:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
aws-ecs-1-nvidia/x86_64/latest/image_id/description?region=region#
```

- For the 64 bit Arm (`arm64`) architecture:

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/
aws-ecs-1-nvidia/arm64/latest/image_id/description?region=region#
```

Next steps

For a detailed tutorial on how to get started with the Bottlerocket operating system on Amazon ECS, see Using a Bottlerocket AMI with Amazon ECS on GitHub and Getting started withBottlerocket and Amazon ECS on the AWS blog site.

For information about how to launch a Bottlerocket instance, see Launching a Bottlerocket instance for Amazon ECS

Launching a Bottlerocket instance for Amazon ECS

You can launch a Bottlerocket instance so that you can run your container workloads.

You can use the AWS CLI to launch the Bottlerocket instance.

1. Create a file that's called `userdata.toml`. This file is used for the instance user data. Replace *`cluster-name`* with the name of your cluster.

   ```
   [settings.ecs]
   cluster = "cluster-name"
   ```

2. Use one of the commands that are included in the section called "Retrieving Amazon ECS-optimized Bottlerocket AMI metadata" to get the Bottlerocket AMI ID. You use this in the following step.
3. Run the following command to launch the Bottlerocket instance. Remember to replace the following parameters:

   - Replace *`subnet`* with the ID of the private or public subnet that your instance will launch in.
   - Replace *`bottlerocket_ami`* with the AMI ID from the previous step.
   - Replace *`t3.large`* with the instance type that you want to use.
   - Replace *`region`* with the Region code.

   ```
   aws ec2 run-instances --key-name ecs-bottlerocket-example \
   ```

```
    --subnet-id subnet \
    --image-id bottlerocket_ami \
    --instance-type t3.large \
    --region region \
    --tag-specifications
 'ResourceType=instance,Tags=[{Key=bottlerocket,Value=example}]' \
    --user-data file://userdata.toml \
    --iam-instance-profile Name=ecsInstanceRole
```

4. Run the following command to verify that the container instance is registered to the cluster. When you run this command, remember to replace the following parameters:

   - Replace *cluster* with your cluster name.
   - Replace *region* with your Region code.

```
aws ecs list-container-instances --cluster cluster-name --region region
```

For a detailed walkthrough of how to get started with the Bottlerocket operating system on Amazon ECS, see Using a Bottlerocket AMI with Amazon ECS on GitHub and Getting started with Bottlerocket and Amazon ECS on the AWS blog site.

Amazon ECS Linux container instance management

When you use EC2 instances for your Amazon ECS workloads, you are responsible for maintaing the instances

Management procedures

- Launching an Amazon ECS Linux container instance
- Bootstrapping Amazon ECS Linux container instances to pass data
- Configuring Amazon ECS Linux container instances to receive Spot Instance notices
- Running a script when you launch an Amazon ECS Linux container instance
- Increasing Amazon ECS Linux container instance network interfaces
- Reserving Amazon ECS Linux container instance memory
- Managing Amazon ECS container instances remotely using AWS Systems Manager
- Using an HTTP proxy for Amazon ECS Linux container instances
- Configuring pre-initialized instances for your Amazon ECS Auto Scaling group

- Updating the Amazon ECS container agent

Each Amazon ECS container agent version supports a different feature set and provides bug fixes from previous versions. When possible, we always recommend using the latest version of the Amazon ECS container agent. To update your container agent to the latest version, see Updating the Amazon ECS container agent.

To see which features and enhancements are included with each agent release, see https://github.com/aws/amazon-ecs-agent/releases.

> **Important**
>
> The minimum Docker version for reliable metrics is Docker version `v20.10.13` and newer, which is included in Amazon ECS-optimized AMI `20220607` and newer.
> Amazon ECS agent versions `1.20.0` and newer have deprecated support for Docker versions older than `18.01.0`.

Launching an Amazon ECS Linux container instance

You can create Amazon ECS container instances using the Amazon EC2 console.

You can launch an instance by various methods including the Amazon EC2 console, AWS CLI, and SDK. For information about the other methods for launching an instance, see Launch your instance in the *Amazon EC2 User Guide*.

For more information about the launch wizard, see Launch an instance using the new launch instance wizard in the *Amazon EC2 User Guide*.

Before you begin, complete the steps in Set up to use Amazon ECS.

You can use the new Amazon EC2 wizard to launch an instance. The launch instance wizard specifies the launch parameters that are required for launching an instance.

Parameters for instance configuration

- Procedure
- Name and tags
- Application and OS Images (Amazon Machine Image)

- Instance type
- Key pair (login)
- Network settings
- Configure storage
- Advanced details

Procedure

Before you begin, complete the steps in Set up to use Amazon ECS.

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation bar at the top of the screen, the current AWS Region is displayed (for example, US East (Ohio)). Select a Region in which to launch the instance.
3. From the Amazon EC2 console dashboard, choose **Launch instance**.

Name and tags

The instance name is a tag, where the key is **Name**, and the value is the name that you specify. You can tag the instance, the volumes, and elastic graphics. For Spot Instances, you can tag the Spot Instance request only.

Specifying an instance name and additional tags is optional.

- For **Name**, enter a descriptive name for the instance. If you don't specify a name, the instance can be identified by its ID, which is automatically generated when you launch the instance.
- To add additional tags, choose **Add additional tags**. Choose **Add tag**, and then enter a key and value, and select the resource type to tag. Choose **Add tag** again for each additional tag to add.

Application and OS Images (Amazon Machine Image)

An Amazon Machine Image (AMI) contains the information required to create an instance. For example, an AMI might contain the software that's required to act as a web server, such as Apache, and your website.

Use the **Search** bar to find a suitable Amazon ECS-optimized AMI published by AWS.

1. Enter one of the following terms in the **Search** bar.

- **`ami-ecs`**
- The **Value** of an Amazon ECS-optimized AMI.

  For the latest Amazon ECS-optimized AMIs and their values, see Linux Amazon ECS-optimized AMI.

2. Press **Enter**.
3. On the **Choose an Amazon Machine Image (AMI)** page, select the **AWS Marketplace AMIs** tab.
4. From the left **Refine results** pane, select **Amazon Web Services** as the **Publisher**.
5. Choose **Select** on the row of the AMI that you want to use.

   Alternatively, choose **Cancel** (at top right) to return to the launch instance wizard without choosing an AMI. A default AMI will be selected. Ensure that the AMI meets the requirements outlined in Amazon ECS-optimized Linux AMIs.

Instance type

The instance type defines the hardware configuration and size of the instance. Larger instance types have more CPU and memory. For more information, see Instance types.

- For **Instance type**, select the instance type for the instance.

  The instance type that you select determines the resources available for your tasks to run on.

Key pair (login)

For **Key pair name**, choose an existing key pair, or choose **Create new key pair** to create a new one.

> **Important**
>
> If you choose the **Proceed without key pair (Not recommended)** option, you won't be able to connect to the instance unless you choose an AMI that is configured to allow users another way to log in.

Network settings

Configure the network settings, as necessary.

- **Networking platform**: Choose **Virtual Private Cloud (VPC)**, and then specify the subnet in the **Network interfaces** section.
- **VPC**: Select an existing VPC in which to create the security group.
- **Subnet**: You can launch an instance in a subnet associated with an Availability Zone, Local Zone, Wavelength Zone, or Outpost.

  To launch the instance in an Availability Zone, select the subnet in which to launch your instance. To create a new subnet, choose **Create new subnet** to go to the Amazon VPC console. When you are done, return to the launch instance wizard and choose the Refresh icon to load your subnet in the list.

  To launch the instance in a Local Zone, select a subnet that you created in the Local Zone.

  To launch an instance in an Outpost, select a subnet in a VPC that you associated with the Outpost.
- **Auto-assign Public IP**: If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

  > **Note**
  >
  > Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.
  > For more information about interface VPC endpoints, see Amazon ECS interface VPC endpoints (AWS PrivateLink)
  > If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see NAT gateways in the *Amazon VPC User Guide* and Using an HTTP proxy for Amazon ECS Linux container instances in this guide.
- **Firewall (security groups)**: Use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored.
  - To select an existing security group, choose **Select existing security group**, and select the security group that you created in Set up to use Amazon ECS.

Configure storage

The AMI you selected includes one or more volumes of storage, including the root volume. You can specify additional volumes to attach to the instance.

You can use the **Simple** view.

- **Storage type**: Configure the storage for your container instance.

  If you are using the Amazon ECS-optimized Amazon Linux 2 AMI, your instance has a single 30 GiB volume configured, which is shared between the operating system and Docker.

  If you are using the Amazon ECS-optimized AMI, your instance has two volumes configured. The **Root** volume is for the operating system's use, and the second Amazon EBS volume (attached to `/dev/xvdcz`) is for Docker's use.

  You can optionally increase or decrease the volume sizes for your instance to meet your application needs.

Advanced details

For **Advanced details**, expand the section to view the fields and specify any additional parameters for the instance.

- **Purchasing option**: Choose **Request Spot Instances** to request Spot Instances. You also need to set the other fields related to Spot Instances. For more information, see Spot Instance Requests.

  > **Note**
  >
  > If you are using Spot Instances and see a `Not available` message, you may need to choose a different instance type.

  .

- **IAM instance profile**: Select your container instance IAM role. This is usually named `ecsInstanceRole`.

> **Important**
>
> If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see Amazon ECS container instance IAM role.

- **User data**: Configure your Amazon ECS container instance with user data, such as the agent environment variables from Amazon ECS container agent configuration. Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:
  - By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing *your_cluster_name* with the name of your cluster.

    ```
    #!/bin/bash
    echo ECS_CLUSTER=your_cluster_name >> /etc/ecs/ecs.config
    ```
  - If you have an `ecs.config` file in Amazon S3 and have enabled Amazon S3 read-only access to your container instance role, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing *your_bucket_name* with the name of your bucket to install the AWS CLI and write your configuration file at launch time.

    > **Note**
    >
    > For more information about this configuration, see Storing Amazon ECS container instance configuration in Amazon S3.

    ```
    #!/bin/bash
    yum install -y aws-cli
    aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
    ```
  - Specify tags for your container instance using the `ECS_CONTAINER_INSTANCE_TAGS` configuration parameter. This creates tags that are associated with Amazon ECS only, they cannot be listed using the Amazon EC2 API.

> **Important**
>
> If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the ECS_CONTAINER_INSTANCE_TAGS agent configuration parameter to add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- Specify tags for your container instance and then use the `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` configuration parameter to propagate them from Amazon EC2 to Amazon ECS

  The following is an example of a user data script that would propagate the tags associated with a container instance, as well as register the container instance with a cluster named `your_cluster_name`:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

For more information, see Bootstrapping Amazon ECS Linux container instances to pass data.

Bootstrapping Amazon ECS Linux container instances to pass data

When you launch an Amazon EC2 instance, you can pass user data to the EC2 instance. The data can be used to perform common automated configuration tasks and even run scripts when the instance boots. For Amazon ECS, the most common use cases for user data are to pass configuration information to the Docker daemon and the Amazon ECS container agent.

You can pass multiple types of user data to Amazon EC2, including cloud boothooks, shell scripts, and `cloud-init` directives. For more information about these and other format types, see the [Cloud-Init documentation](#).

To pass the user data when using the Amazon EC2 launch wizard, see [Launching an Amazon ECS Linux container instance](#).

You can configure the container instance to pass data in the container agent configuration or in the Docker daemon configuration.

Amazon ECS container agent

The Linux variants of the Amazon ECS-optimized AMI look for agent configuration data in the `/etc/ecs/ecs.config` file when the container agent starts. You can specify this configuration data at launch with Amazon EC2 user data. For more information about available Amazon ECS container agent configuration variables, see [Amazon ECS container agent configuration](#).

To set only a single agent configuration variable, such as the cluster name, use **echo** to copy the variable to the configuration file:

```
#!/bin/bash
echo "ECS_CLUSTER=MyCluster" >> /etc/ecs/ecs.config
```

If you have multiple variables to write to `/etc/ecs/ecs.config`, use the following `heredoc` format. This format writes everything between the lines beginning with **cat** and `EOF` to the configuration file.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
ECS_LOGLEVEL=debug
ECS_WARM_POOLS_CHECK=true
EOF
```

To set custom instance attributes, set the `ECS_INSTANCE_ATTRIBUTES` environment variable.

```
#!/bin/bash
```

```
cat <<'EOF' >> ecs.config
ECS_INSTANCE_ATTRIBUTES={"envtype":"prod"}
EOF
```

Docker daemon

You can specify Docker daemon configuration information with Amazon EC2 user data. For more information about configuration options, see the Docker daemon documentation.

> **Note**
>
> AWS doesn't support custom Docker configurations, because they can sometimes conflict with future Amazon ECS changes or features without warning.

In the example below, the custom options are added to the Docker daemon configuration file, `/etc/docker/daemon.json` which is then specified in the user data when the instance is launched.

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"debug": true}
EOF
systemctl restart docker --no-block
```

In the example below, the custom options are added to the Docker daemon configuration file, `/etc/docker/daemon.json` which is then specified in the user data when the instance is launched. This example shows how to disable the docker-proxy in the Docker daemon config file.

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"userland-proxy": false}
EOF
systemctl restart docker --no-block
```

Configuring Amazon ECS Linux container instances to receive Spot Instance notices

Amazon EC2 terminates, stops, or hibernates your Spot Instance when the Spot price exceeds the maximum price for your request or capacity is no longer available. Amazon EC2 provides a Spot Instance two-minute interruption notice for terminate and stop actions. It does not provide the two-minute notice for the hibernate action. If Amazon ECS Spot Instance draining is turned onon the instance, Amazon ECS receives the Spot Instance interruption notice and places the instance in `DRAINING` status.

> **Important**
>
> Amazon ECS does not receive a notice from Amazon EC2 when instances are removed by Auto Scaling Capacity Rebalancing. For more information, see Amazon EC2 Auto Scaling Capacity Rebalancing.

When a container instance is set to `DRAINING`, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Service tasks on the draining container instance that are in the `PENDING` state are stopped immediately. If there are container instances in the cluster that are available, replacement service tasks are started on them.

Spot Instance draining is turned off by default.

You can turn on Spot Instance draining when you launch an instance. Add the following script into the **User data** field. Replace *`MyCluster`* with the name of the cluster to register the container instance to.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
EOF
```

For more information, see Launching an Amazon ECS Linux container instance.

To turn on Spot Instance draining for an existing container instance

1. Connect to the Spot Instance over SSH.
2. Edit the `/etc/ecs/ecs.config` file and add the following:

   ```
   ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
   ```

3. Restart the `ecs` service.

   - For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

4. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see [the section called “Container introspection”](#).

```
curl http://localhost:51678/v1/metadata
```

Running a script when you launch an Amazon ECS Linux container instance

You might need to run a specific container on every container instance to deal with operations or security concerns such as monitoring, security, metrics, service discovery, or logging.

To do this, you can configure your container instances to call the **docker run** command with the user data script at launch, or in some init system such as Upstart or **systemd**. While this method works, it has some disadvantages because Amazon ECS has no knowledge of the container and cannot monitor the CPU, memory, ports, or any other resources used. To ensure that Amazon ECS can properly account for all task resources, create a task definition for the container to run on your container instances. Then, use Amazon ECS to place the task at launch time with Amazon EC2 user data.

The Amazon EC2 user data script in the following procedure uses the Amazon ECS introspection API to identify the container instance. Then, it uses the AWS CLI and the **start-task** command to run a specified task on itself during startup.

To start a task at container instance launch time

1. Modify your `ecsInstanceRole` IAM role to add permissions for the `StartTask` API operation. For more information, see [Update permissions for a role](#) in the *AWS Identity and Access Management User Guide*.
2. Launch one or more container instances using the Amazon ECS-optimized Amazon Linux 2 AMI. Launch new container instances and use the following example script in the EC2 User data. Replace *`your_cluster_name`* with the cluster for the container instance to register into and *`my_task_def`* with the task definition to run on the instance at launch.

   For more information, see [Launching an Amazon ECS Linux container instance](#).

> **Note**
>
> The MIME multi-part content below uses a shell script to set configuration values and install packages. It also uses a systemd job to start the task after the **ecs** service is running and the introspection API is available.

```
Content-Type: multipart/mixed; boundary="==BOUNDARY=="
MIME-Version: 1.0

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
# Specify the cluster that the container instance should register into
cluster=your_cluster_name

# Write the cluster configuration variable to the ecs.config file
# (add any other configuration variables here also)
echo ECS_CLUSTER=$cluster >> /etc/ecs/ecs.config

START_TASK_SCRIPT_FILE="/etc/ecs/ecs-start-task.sh"
cat <<- 'EOF' > ${START_TASK_SCRIPT_FILE}
 exec 2>>/var/log/ecs/ecs-start-task.log
 set -x

 # Install prerequisite tools
 yum install -y jq aws-cli

 # Wait for the ECS service to be responsive
 until curl -s http://localhost:51678/v1/metadata
 do
  sleep 1
 done

 # Grab the container instance ARN and AWS Region from instance metadata
 instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F/ '{print $NF}' )
 cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '. | .Cluster' | awk
 -F/ '{print $NF}' )
```

```
 region=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F: '{print $4}')

 # Specify the task definition to run at launch
 task_definition=my_task_def

 # Run the AWS CLI start-task command to start your task on this container instance
 aws ecs start-task --cluster $cluster --task-definition $task_definition --
container-instances $instance_arn --started-by $instance_arn --region $region
EOF

# Write systemd unit file
UNIT="ecs-start-task.service"
cat <<- EOF > /etc/systemd/system/${UNIT}
      [Unit]
      Description=ECS Start Task
      Requires=ecs.service
      After=ecs.service

      [Service]
      Restart=on-failure
      RestartSec=30
      ExecStart=/usr/bin/bash ${START_TASK_SCRIPT_FILE}

      [Install]
      WantedBy=default.target
EOF

# Enable our ecs.service dependent service with `--no-block` to prevent systemd
 deadlock
# See https://github.com/aws/amazon-ecs-agent/issues/1707
systemctl enable --now --no-block "${UNIT}"
--==BOUNDARY==--
```

3. Verify that your container instances launch into the correct cluster and that your tasks have started.

   a. Open the console at https://console.aws.amazon.com/ecs/v2.

   b. From the navigation bar, choose the Region that your cluster is in.

   c. In the navigation pane, choose **Clusters** and select the cluster that hosts your container instances.

   d. On the **Cluster** page, choose **Tasks**, and then choose yor tasks.

Each container instance you launched should have your task running on it.

If you do not see your tasks, you can log in to your container instances with SSH and check the `/var/log/ecs/ecs-start-task.log` file for debugging information.

Increasing Amazon ECS Linux container instance network interfaces

> **Note**
>
> This feature is not available on Fargate.

Each task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the container instance that hosts it. There is a default limit to the number of network interfaces that can be attached to an Amazon EC2 instance, and the primary network interface counts as one. For example, by default a `c5.large` instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs to the instance. Because each task using the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type.

Amazon ECS supports launching container instances with increased ENI density using supported Amazon EC2 instance types. When you use these instance types and turn on the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks on each container instance. To use the console to turn on the feature, see Modifying Amazon ECS account settings. To use the AWS CLI to turn on the feature, see Managing Amazon ECS account settings using the AWS CLI.

For example, a `c5.large` instance with `awsvpcTrunking` has an increased ENI limit of twelve. The container instance will have the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the container instance. So this configuration allows you to launch ten tasks on the container instance instead of the current two tasks.

The trunk network interface is fully managed by Amazon ECS and is deleted when you either terminate or deregister your container instance from the cluster. For more information, see Amazon ECS task networking options for the EC2 launch type.

Considerations

Consider the following when using the ENI trunking feature.

- Only Linux variants of the Amazon ECS-optimized AMI, or other Amazon Linux variants with version `1.28.1` or later of the container agent and version `1.28.1-2` or later of the ecs-init package, support the increased ENI limits. If you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. Windows containers are not supported at this time.
- Only new Amazon EC2 instances launched after enabling `awsvpcTrunking` receive the increased ENI limits and the trunk network interface. Previously launched instances do not receive these features regardless of the actions taken.
- Amazon EC2 instances must have resource-based IPv4 DNS requests turned off. To disable this option, clear the **Enable resource-based IPV4 (A record) DNS requests** option when you create a new instance in the Amazon EC2 console. To disable this option using the AWS CLI, use the following command.

```
aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxx --no-enable-resource-name-dns-a-record --no-dry-run
```

- Amazon EC2 instances in shared subnets are not supported. They will fail to register to a cluster if they are used.
- Your tasks must use the `awsvpc` network mode and the EC2 launch type. Tasks using the Fargate launch type always received a dedicated ENI regardless of how many are launched, so this feature is not needed.
- Your tasks must be launched in the same Amazon VPC as your container instance. Your tasks will fail to start with an attribute error if they are not within the same VPC.
- When launching a new container instance, the instance transitions to a `REGISTERING` status while the trunk elastic network interface is provisioned for the instance. If the registration fails, the instance transitions to a `REGISTRATION_FAILED` status. You can troubleshoot a failed registration by describing the container instance to view the `statusReason` field which describes the reason for the failure. The container instance then can be manually deregistered or terminated. Once the container instance is successfully deregistered or terminated, Amazon ECS will delete the trunk ENI.

> **Note**
>
> Amazon ECS emits container instance state change events which you can monitor for instances that transition to a `REGISTRATION_FAILED` state. For more information, see Amazon ECS container instance state change events.

- Once the container instance is terminated, the instance transitions to a `DEREGISTERING` status while the trunk elastic network interface is deprovisioned. The instance then transitions to an `INACTIVE` status.
- If a container instance in a public subnet with the increased ENI limits is stopped and then restarted, the instance loses its public IP address, and the container agent loses its connection.
- When you enable `awsvpcTrunking`, container instances receive an additional ENI that uses the VPC's default security group, and is managed by Amazon ECS.

  A default VPC comes with a public subnet in each Availability Zone, an internet gateway, and settings to enable DNS resolution. The subnet is a public subnet, because the main route table sends the subnet's traffic that is destined for the internet to the internet gateway. You can make a default subnet into a private subnet by removing the route from the destination 0.0.0.0/0 to the internet gateway. However, if you do this, no container instance running in that subnet can access the internet. You can add or delete security group rules to control the traffic into and out of your subnets. For more information, see Security group rules in the *Amazon Virtual Private Cloud User Guide*.

Prerequisites

Before you launch a container instance with the increased ENI limits, the following prerequisites must be completed.

- The service-linked role for Amazon ECS must be created. The Amazon ECS service-linked role provides Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster, or if you create or update a service in the AWS Management Console. For more information, see Using service-linked roles for Amazon ECS. You can also create the service-linked role with the following AWS CLI command.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Your account or container instance IAM role must enable the `awsvpcTrunking` account setting. We recommend that you create 2 container instance roles (`ecsInstanceRole`). You can then enable the `awsvpcTrunking` account setting for one role and use that role for tasks that require ENI trunking. For information about the container instance role, see Amazon ECS container instance IAM role.

After the prerequisites are met, you can launch a new container instance using one of the supported Amazon EC2 instance types, and the instance will have the increased ENI limits. For a list of supported instance types, see Supported instances for increased Amazon ECS container network interfaces. The container instance must have version `1.28.1` or later of the container agent and version `1.28.1-2` or later of the ecs-init package. If you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. For more information, see Launching an Amazon ECS Linux container instance.

> **Important**
>
> Amazon EC2 instances must have resource-based IPv4 DNS requests turned off. To disable this option, ensure the **Enable resource-based IPV4 (A record) DNS requests** option is deselected when creating a new instance using the Amazon EC2 console. To disable this option using the AWS CLI, use the following command.
>
> ```
> aws ec2 modify-private-dns-name-options --instance-id i-xxxxxxx --no-enable-resource-name-dns-a-record --no-dry-run
> ```

To view your container instances with increased ENI limits with the AWS CLI

Each container instance has a default network interface, referred to as a trunk network interface. Use the following command to list your container instances with increased ENI limits by querying for the `ecs.awsvpc-trunk-id` attribute, which indicates it has a trunk network interface.

- list-attributes (AWS CLI)

```
aws ecs list-attributes \
      --target-type container-instance \
      --attribute-name ecs.awsvpc-trunk-id \
      --cluster cluster_name \
      --region us-east-1
```

- Get-ECSAttributeList (AWS Tools for Windows PowerShell)

```
Get-ECSAttributeList -TargetType container-instance -AttributeName ecs.awsvpc-trunk-id -Region us-east-1
```

Supported instances for increased Amazon ECS container network interfaces

The following shows the supported Amazon EC2 instance types and how many tasks using the `awsvpc` network mode can be launched on each instance type before and after enabling the `awsvpcTrunking` account setting.

> **Important**
>
> Although other instance types are supported in the same instance family, the `a1.metal`, `c5.metal`, `c5a.8xlarge`, `c5ad.8xlarge`, `c5d.metal`, `m5.metal`, `p3dn.24xlarge`, `r5.metal`, `r5.8xlarge`, and `r5d.metal` instance types are not supported.
> The `c5n`, `d3`, `d3en`, `g3`, `g3s`, `g4dn`, `i3`, `i3en`, `inf1`, `m5dn`, `m5n`, `m5zn`, `mac1`, `r5b`, `r5n`, `r5dn`, `u-12tb1`, `u-6tb1`, `u-9tb1`, and `z1d` instance families are not supported.

Topics

- General purpose
- Compute optimized
- Memory optimized
- Storage optimized
- Accelerated computing
- High performance computing

General purpose

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
a1.medium	1	10
a1.large	2	10
a1.xlarge	3	20
a1.2xlarge	3	40
a1.4xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m5.large	2	10
m5.xlarge	3	20
m5.2xlarge	3	40
m5.4xlarge	7	60
m5.8xlarge	7	60
m5.12xlarge	7	60
m5.16xlarge	14	120
m5.24xlarge	14	120
m5a.large	2	10
m5a.xlarge	3	20
m5a.2xlarge	3	40
m5a.4xlarge	7	60
m5a.8xlarge	7	60
m5a.12xlarge	7	60
m5a.16xlarge	14	120
m5a.24xlarge	14	120
m5ad.large	2	10
m5ad.xlarge	3	20
m5ad.2xlarge	3	40
m5ad.4xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m5ad.8xlarge	7	60
m5ad.12xlarge	7	60
m5ad.16xlarge	14	120
m5ad.24xlarge	14	120
m5d.large	2	10
m5d.xlarge	3	20
m5d.2xlarge	3	40
m5d.4xlarge	7	60
m5d.8xlarge	7	60
m5d.12xlarge	7	60
m5d.16xlarge	14	120
m5d.24xlarge	14	120
m5d.metal	14	120
m6a.large	2	10
m6a.xlarge	3	20
m6a.2xlarge	3	40
m6a.4xlarge	7	60
m6a.8xlarge	7	90
m6a.12xlarge	7	120
m6a.16xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m6a.24xlarge	14	120
m6a.32xlarge	14	120
m6a.48xlarge	14	120
m6a.metal	14	120
m6g.medium	1	4
m6g.large	2	10
m6g.xlarge	3	20
m6g.2xlarge	3	40
m6g.4xlarge	7	60
m6g.8xlarge	7	60
m6g.12xlarge	7	60
m6g.16xlarge	14	120
m6g.metal	14	120
m6gd.medium	1	4
m6gd.large	2	10
m6gd.xlarge	3	20
m6gd.2xlarge	3	40
m6gd.4xlarge	7	60
m6gd.8xlarge	7	60
m6gd.12xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m6gd.16xlarge	14	120
m6gd.metal	14	120
m6i.large	2	10
m6i.xlarge	3	20
m6i.2xlarge	3	40
m6i.4xlarge	7	60
m6i.8xlarge	7	90
m6i.12xlarge	7	120
m6i.16xlarge	14	120
m6i.24xlarge	14	120
m6i.32xlarge	14	120
m6i.metal	14	120
m6id.large	2	10
m6id.xlarge	3	20
m6id.2xlarge	3	40
m6id.4xlarge	7	60
m6id.8xlarge	7	90
m6id.12xlarge	7	120
m6id.16xlarge	14	120
m6id.24xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m6id.32xlarge	14	120
m6id.metal	14	120
m6idn.large	2	10
m6idn.xlarge	3	20
m6idn.2xlarge	3	40
m6idn.4xlarge	7	60
m6idn.8xlarge	7	90
m6idn.12xlarge	7	120
m6idn.16xlarge	14	120
m6idn.24xlarge	14	120
m6idn.32xlarge	15	120
m6idn.metal	15	120
m6in.large	2	10
m6in.xlarge	3	20
m6in.2xlarge	3	40
m6in.4xlarge	7	60
m6in.8xlarge	7	90
m6in.12xlarge	7	120
m6in.16xlarge	14	120
m6in.24xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m6in.32xlarge	15	120
m6in.metal	15	120
m7a.medium	1	4
m7a.large	2	10
m7a.xlarge	3	20
m7a.2xlarge	3	40
m7a.4xlarge	7	60
m7a.8xlarge	7	90
m7a.12xlarge	7	120
m7a.16xlarge	14	120
m7a.24xlarge	14	120
m7a.32xlarge	14	120
m7a.48xlarge	14	120
m7a.metal-48xl	14	120
m7g.medium	1	4
m7g.large	2	10
m7g.xlarge	3	20
m7g.2xlarge	3	40
m7g.4xlarge	7	60
m7g.8xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m7g.12xlarge	7	60
m7g.16xlarge	14	120
m7g.metal	14	120
m7gd.medium	1	4
m7gd.large	2	10
m7gd.xlarge	3	20
m7gd.2xlarge	3	40
m7gd.4xlarge	7	60
m7gd.8xlarge	7	60
m7gd.12xlarge	7	60
m7gd.16xlarge	14	120
m7gd.metal	14	120
m7i.large	2	10
m7i.xlarge	3	20
m7i.2xlarge	3	40
m7i.4xlarge	7	60
m7i.8xlarge	7	90
m7i.12xlarge	7	120
m7i.16xlarge	14	120
m7i.24xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m7i.48xlarge	14	120
m7i.metal-24xl	14	120
m7i.metal-48xl	14	120
m7i-flex.large	2	4
m7i-flex.xlarge	3	10
m7i-flex.2xlarge	3	20
m7i-flex.4xlarge	7	40
m7i-flex.8xlarge	7	60
m7i-flex.12xlarge	7	120
m7i-flex.16xlarge	14	120
m8g.medium	1	4
m8g.large	2	10
m8g.xlarge	3	20
m8g.2xlarge	3	40
m8g.4xlarge	7	60
m8g.8xlarge	7	60
m8g.12xlarge	7	60
m8g.16xlarge	14	120
m8g.24xlarge	14	120
m8g.48xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m8g.metal-24xl	14	120
m8g.metal-48xl	14	120
m8gd.medium	1	4
m8gd.large	2	10
m8gd.xlarge	3	20
m8gd.2xlarge	3	40
m8gd.4xlarge	7	60
m8gd.8xlarge	7	60
m8gd.12xlarge	7	60
m8gd.16xlarge	14	120
m8gd.24xlarge	14	120
m8gd.48xlarge	14	120
m8gd.metal-24xl	14	120
m8gd.metal-48xl	14	120
m8i.large	2	10
m8i.xlarge	3	20
m8i.2xlarge	3	40
m8i.4xlarge	7	60
m8i.8xlarge	9	90
m8i.12xlarge	11	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
m8i.16xlarge	15	120
m8i.24xlarge	15	120
m8i.32xlarge	23	120
m8i.48xlarge	23	120
m8i.96xlarge	23	120
m8i.metal-48xl	23	120
m8i.metal-96xl	23	120
m8i-flex.large	2	4
m8i-flex.xlarge	3	10
m8i-flex.2xlarge	3	20
m8i-flex.4xlarge	7	40
m8i-flex.8xlarge	9	60
m8i-flex.12xlarge	11	120
m8i-flex.16xlarge	15	120
mac2.metal	7	12
mac2-m1ultra.metal	7	12
mac2-m2.metal	7	12
mac2-m2pro.metal	7	12

Compute optimized

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c5.large	2	10
c5.xlarge	3	20
c5.2xlarge	3	40
c5.4xlarge	7	60
c5.9xlarge	7	60
c5.12xlarge	7	60
c5.18xlarge	14	120
c5.24xlarge	14	120
c5a.large	2	10
c5a.xlarge	3	20
c5a.2xlarge	3	40
c5a.4xlarge	7	60
c5a.12xlarge	7	60
c5a.16xlarge	14	120
c5a.24xlarge	14	120
c5ad.large	2	10
c5ad.xlarge	3	20
c5ad.2xlarge	3	40
c5ad.4xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c5ad.12xlarge	7	60
c5ad.16xlarge	14	120
c5ad.24xlarge	14	120
c5d.large	2	10
c5d.xlarge	3	20
c5d.2xlarge	3	40
c5d.4xlarge	7	60
c5d.9xlarge	7	60
c5d.12xlarge	7	60
c5d.18xlarge	14	120
c5d.24xlarge	14	120
c6a.large	2	10
c6a.xlarge	3	20
c6a.2xlarge	3	40
c6a.4xlarge	7	60
c6a.8xlarge	7	90
c6a.12xlarge	7	120
c6a.16xlarge	14	120
c6a.24xlarge	14	120
c6a.32xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c6a.48xlarge	14	120
c6a.metal	14	120
c6g.medium	1	4
c6g.large	2	10
c6g.xlarge	3	20
c6g.2xlarge	3	40
c6g.4xlarge	7	60
c6g.8xlarge	7	60
c6g.12xlarge	7	60
c6g.16xlarge	14	120
c6g.metal	14	120
c6gd.medium	1	4
c6gd.large	2	10
c6gd.xlarge	3	20
c6gd.2xlarge	3	40
c6gd.4xlarge	7	60
c6gd.8xlarge	7	60
c6gd.12xlarge	7	60
c6gd.16xlarge	14	120
c6gd.metal	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c6gn.medium	1	4
c6gn.large	2	10
c6gn.xlarge	3	20
c6gn.2xlarge	3	40
c6gn.4xlarge	7	60
c6gn.8xlarge	7	60
c6gn.12xlarge	7	60
c6gn.16xlarge	14	120
c6i.large	2	10
c6i.xlarge	3	20
c6i.2xlarge	3	40
c6i.4xlarge	7	60
c6i.8xlarge	7	90
c6i.12xlarge	7	120
c6i.16xlarge	14	120
c6i.24xlarge	14	120
c6i.32xlarge	14	120
c6i.metal	14	120
c6id.large	2	10
c6id.xlarge	3	20

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c6id.2xlarge	3	40
c6id.4xlarge	7	60
c6id.8xlarge	7	90
c6id.12xlarge	7	120
c6id.16xlarge	14	120
c6id.24xlarge	14	120
c6id.32xlarge	14	120
c6id.metal	14	120
c6in.large	2	10
c6in.xlarge	3	20
c6in.2xlarge	3	40
c6in.4xlarge	7	60
c6in.8xlarge	7	90
c6in.12xlarge	7	120
c6in.16xlarge	14	120
c6in.24xlarge	14	120
c6in.32xlarge	15	120
c6in.metal	15	120
c7a.medium	1	4
c7a.large	2	10

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c7a.xlarge	3	20
c7a.2xlarge	3	40
c7a.4xlarge	7	60
c7a.8xlarge	7	90
c7a.12xlarge	7	120
c7a.16xlarge	14	120
c7a.24xlarge	14	120
c7a.32xlarge	14	120
c7a.48xlarge	14	120
c7a.metal-48xl	14	120
c7g.medium	1	4
c7g.large	2	10
c7g.xlarge	3	20
c7g.2xlarge	3	40
c7g.4xlarge	7	60
c7g.8xlarge	7	60
c7g.12xlarge	7	60
c7g.16xlarge	14	120
c7g.metal	14	120
c7gd.medium	1	4

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c7gd.large	2	10
c7gd.xlarge	3	20
c7gd.2xlarge	3	40
c7gd.4xlarge	7	60
c7gd.8xlarge	7	60
c7gd.12xlarge	7	60
c7gd.16xlarge	14	120
c7gd.metal	14	120
c7gn.medium	1	4
c7gn.large	2	10
c7gn.xlarge	3	20
c7gn.2xlarge	3	40
c7gn.4xlarge	7	60
c7gn.8xlarge	7	60
c7gn.12xlarge	7	60
c7gn.16xlarge	14	120
c7gn.metal	14	120
c7i.large	2	10
c7i.xlarge	3	20
c7i.2xlarge	3	40

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c7i.4xlarge	7	60
c7i.8xlarge	7	90
c7i.12xlarge	7	120
c7i.16xlarge	14	120
c7i.24xlarge	14	120
c7i.48xlarge	14	120
c7i.metal-24xl	14	120
c7i.metal-48xl	14	120
c7i-flex.large	2	4
c7i-flex.xlarge	3	10
c7i-flex.2xlarge	3	20
c7i-flex.4xlarge	7	40
c7i-flex.8xlarge	7	60
c7i-flex.12xlarge	7	120
c7i-flex.16xlarge	14	120
c8g.medium	1	4
c8g.large	2	10
c8g.xlarge	3	20
c8g.2xlarge	3	40
c8g.4xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c8g.8xlarge	7	60
c8g.12xlarge	7	60
c8g.16xlarge	14	120
c8g.24xlarge	14	120
c8g.48xlarge	14	120
c8g.metal-24xl	14	120
c8g.metal-48xl	14	120
c8gd.medium	1	4
c8gd.large	2	10
c8gd.xlarge	3	20
c8gd.2xlarge	3	40
c8gd.4xlarge	7	60
c8gd.8xlarge	7	60
c8gd.12xlarge	7	60
c8gd.16xlarge	14	120
c8gd.24xlarge	14	120
c8gd.48xlarge	14	120
c8gd.metal-24xl	14	120
c8gd.metal-48xl	14	120
c8gn.medium	1	4

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
c8gn.large	2	10
c8gn.xlarge	3	20
c8gn.2xlarge	3	40
c8gn.4xlarge	7	60
c8gn.8xlarge	9	60
c8gn.12xlarge	11	60
c8gn.16xlarge	15	120
c8gn.24xlarge	23	120
c8gn.48xlarge	23	120
c8gn.metal-24xl	23	120
c8gn.metal-48xl	23	120

Memory optimized

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r5.large	2	10
r5.xlarge	3	20
r5.2xlarge	3	40
r5.4xlarge	7	60
r5.12xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r5.16xlarge	14	120
r5.24xlarge	14	120
r5a.large	2	10
r5a.xlarge	3	20
r5a.2xlarge	3	40
r5a.4xlarge	7	60
r5a.8xlarge	7	60
r5a.12xlarge	7	60
r5a.16xlarge	14	120
r5a.24xlarge	14	120
r5ad.large	2	10
r5ad.xlarge	3	20
r5ad.2xlarge	3	40
r5ad.4xlarge	7	60
r5ad.8xlarge	7	60
r5ad.12xlarge	7	60
r5ad.16xlarge	14	120
r5ad.24xlarge	14	120
r5b.16xlarge	14	120
r5d.large	2	10

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r5d.xlarge	3	20
r5d.2xlarge	3	40
r5d.4xlarge	7	60
r5d.8xlarge	7	60
r5d.12xlarge	7	60
r5d.16xlarge	14	120
r5d.24xlarge	14	120
r5dn.16xlarge	14	120
r6a.large	2	10
r6a.xlarge	3	20
r6a.2xlarge	3	40
r6a.4xlarge	7	60
r6a.8xlarge	7	90
r6a.12xlarge	7	120
r6a.16xlarge	14	120
r6a.24xlarge	14	120
r6a.32xlarge	14	120
r6a.48xlarge	14	120
r6a.metal	14	120
r6g.medium	1	4

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r6g.large	2	10
r6g.xlarge	3	20
r6g.2xlarge	3	40
r6g.4xlarge	7	60
r6g.8xlarge	7	60
r6g.12xlarge	7	60
r6g.16xlarge	14	120
r6g.metal	14	120
r6gd.medium	1	4
r6gd.large	2	10
r6gd.xlarge	3	20
r6gd.2xlarge	3	40
r6gd.4xlarge	7	60
r6gd.8xlarge	7	60
r6gd.12xlarge	7	60
r6gd.16xlarge	14	120
r6gd.metal	14	120
r6i.large	2	10
r6i.xlarge	3	20
r6i.2xlarge	3	40

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r6i.4xlarge	7	60
r6i.8xlarge	7	90
r6i.12xlarge	7	120
r6i.16xlarge	14	120
r6i.24xlarge	14	120
r6i.32xlarge	14	120
r6i.metal	14	120
r6idn.large	2	10
r6idn.xlarge	3	20
r6idn.2xlarge	3	40
r6idn.4xlarge	7	60
r6idn.8xlarge	7	90
r6idn.12xlarge	7	120
r6idn.16xlarge	14	120
r6idn.24xlarge	14	120
r6idn.32xlarge	15	120
r6idn.metal	15	120
r6in.large	2	10
r6in.xlarge	3	20
r6in.2xlarge	3	40

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r6in.4xlarge	7	60
r6in.8xlarge	7	90
r6in.12xlarge	7	120
r6in.16xlarge	14	120
r6in.24xlarge	14	120
r6in.32xlarge	15	120
r6in.metal	15	120
r6id.large	2	10
r6id.xlarge	3	20
r6id.2xlarge	3	40
r6id.4xlarge	7	60
r6id.8xlarge	7	90
r6id.12xlarge	7	120
r6id.16xlarge	14	120
r6id.24xlarge	14	120
r6id.32xlarge	14	120
r6id.metal	14	120
r7a.medium	1	4
r7a.large	2	10
r7a.xlarge	3	20

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r7a.2xlarge	3	40
r7a.4xlarge	7	60
r7a.8xlarge	7	90
r7a.12xlarge	7	120
r7a.16xlarge	14	120
r7a.24xlarge	14	120
r7a.32xlarge	14	120
r7a.48xlarge	14	120
r7a.metal-48xl	14	120
r7g.medium	1	4
r7g.large	2	10
r7g.xlarge	3	20
r7g.2xlarge	3	40
r7g.4xlarge	7	60
r7g.8xlarge	7	60
r7g.12xlarge	7	60
r7g.16xlarge	14	120
r7g.metal	14	120
r7gd.medium	1	4
r7gd.large	2	10

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r7gd.xlarge	3	20
r7gd.2xlarge	3	40
r7gd.4xlarge	7	60
r7gd.8xlarge	7	60
r7gd.12xlarge	7	60
r7gd.16xlarge	14	120
r7gd.metal	14	120
r7i.large	2	10
r7i.xlarge	3	20
r7i.2xlarge	3	40
r7i.4xlarge	7	60
r7i.8xlarge	7	90
r7i.12xlarge	7	120
r7i.16xlarge	14	120
r7i.24xlarge	14	120
r7i.48xlarge	14	120
r7i.metal-24xl	14	120
r7i.metal-48xl	14	120
r7iz.large	2	10
r7iz.xlarge	3	20

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r7iz.2xlarge	3	40
r7iz.4xlarge	7	60
r7iz.8xlarge	7	90
r7iz.12xlarge	7	120
r7iz.16xlarge	14	120
r7iz.32xlarge	14	120
r7iz.metal-16xl	14	120
r7iz.metal-32xl	14	120
r8g.medium	1	4
r8g.large	2	10
r8g.xlarge	3	20
r8g.2xlarge	3	40
r8g.4xlarge	7	60
r8g.8xlarge	7	60
r8g.12xlarge	7	60
r8g.16xlarge	14	120
r8g.24xlarge	14	120
r8g.48xlarge	14	120
r8g.metal-24xl	14	120
r8g.metal-48xl	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r8gd.medium	1	4
r8gd.large	2	10
r8gd.xlarge	3	20
r8gd.2xlarge	3	40
r8gd.4xlarge	7	60
r8gd.8xlarge	7	60
r8gd.12xlarge	7	60
r8gd.16xlarge	14	120
r8gd.24xlarge	14	120
r8gd.48xlarge	14	120
r8gd.metal-24xl	14	120
r8gd.metal-48xl	14	120
r8i.large	2	10
r8i.xlarge	3	20
r8i.2xlarge	3	40
r8i.4xlarge	7	60
r8i.8xlarge	9	90
r8i.12xlarge	11	120
r8i.16xlarge	15	120
r8i.24xlarge	15	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
r8i.32xlarge	23	120
r8i.48xlarge	23	120
r8i.96xlarge	23	120
r8i.metal-48xl	23	120
r8i.metal-96xl	23	120
r8i-flex.large	2	4
r8i-flex.xlarge	3	10
r8i-flex.2xlarge	3	20
r8i-flex.4xlarge	7	40
r8i-flex.8xlarge	9	60
r8i-flex.12xlarge	11	120
r8i-flex.16xlarge	15	120
u-3tb1.56xlarge	7	12
u-6tb1.56xlarge	14	12
u-18tb1.112xlarge	14	12
u-18tb1.metal	14	12
u-24tb1.112xlarge	14	12
u-24tb1.metal	14	12
u7i-6tb.112xlarge	14	120
u7i-8tb.112xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
u7i-12tb.224xlarge	14	120
u7in-16tb.224xlarge	15	120
u7in-24tb.224xlarge	15	120
u7in-32tb.224xlarge	15	120
u7inh-32tb.480xlarge	15	120
x2gd.medium	1	10
x2gd.large	2	10
x2gd.xlarge	3	20
x2gd.2xlarge	3	40
x2gd.4xlarge	7	60
x2gd.8xlarge	7	60
x2gd.12xlarge	7	60
x2gd.16xlarge	14	120
x2gd.metal	14	120
x2idn.16xlarge	14	120
x2idn.24xlarge	14	120
x2idn.32xlarge	14	120
x2idn.metal	14	120
x2iedn.xlarge	3	13
x2iedn.2xlarge	3	29

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
x2iedn.4xlarge	7	60
x2iedn.8xlarge	7	120
x2iedn.16xlarge	14	120
x2iedn.24xlarge	14	120
x2iedn.32xlarge	14	120
x2iedn.metal	14	120
x2iezn.2xlarge	3	64
x2iezn.4xlarge	7	120
x2iezn.6xlarge	7	120
x2iezn.8xlarge	7	120
x2iezn.12xlarge	14	120
x2iezn.metal	14	120
x8g.medium	1	4
x8g.large	2	10
x8g.xlarge	3	20
x8g.2xlarge	3	40
x8g.4xlarge	7	60
x8g.8xlarge	7	60
x8g.12xlarge	7	60
x8g.16xlarge	14	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
x8g.24xlarge	14	120
x8g.48xlarge	14	120
x8g.metal-24xl	14	120
x8g.metal-48xl	14	120

Storage optimized

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
i4g.large	2	10
i4g.xlarge	3	20
i4g.2xlarge	3	40
i4g.4xlarge	7	60
i4g.8xlarge	7	60
i4g.16xlarge	14	120
i4i.xlarge	3	8
i4i.2xlarge	3	28
i4i.4xlarge	7	58
i4i.8xlarge	7	118
i4i.12xlarge	7	118
i4i.16xlarge	14	248

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
i4i.24xlarge	14	118
i4i.32xlarge	14	498
i4i.metal	14	498
i7i.large	2	10
i7i.xlarge	3	20
i7i.2xlarge	3	40
i7i.4xlarge	7	60
i7i.8xlarge	7	90
i7i.12xlarge	7	90
i7i.16xlarge	14	120
i7i.24xlarge	14	120
i7i.48xlarge	14	120
i7i.metal-24xl	14	120
i7i.metal-48xl	14	120
i7ie.large	2	20
i7ie.xlarge	3	29
i7ie.2xlarge	3	29
i7ie.3xlarge	3	29
i7ie.6xlarge	7	60
i7ie.12xlarge	7	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
i7ie.18xlarge	14	120
i7ie.24xlarge	14	120
i7ie.48xlarge	14	120
i7ie.metal-24xl	14	120
i7ie.metal-48xl	14	120
i8g.large	2	10
i8g.xlarge	3	20
i8g.2xlarge	3	40
i8g.4xlarge	7	60
i8g.8xlarge	7	60
i8g.12xlarge	7	60
i8g.16xlarge	14	120
i8g.24xlarge	14	120
i8g.48xlarge	14	120
i8g.metal-24xl	14	120
i8ge.large	2	20
i8ge.xlarge	3	29
i8ge.2xlarge	3	29
i8ge.3xlarge	5	29
i8ge.6xlarge	9	60

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
i8ge.12xlarge	11	60
i8ge.18xlarge	15	120
i8ge.24xlarge	15	120
i8ge.48xlarge	23	120
i8ge.metal-24xl	15	120
i8ge.metal-48xl	23	120
im4gn.large	2	10
im4gn.xlarge	3	20
im4gn.2xlarge	3	40
im4gn.4xlarge	7	60
im4gn.8xlarge	7	60
im4gn.16xlarge	14	120
is4gen.medium	1	4
is4gen.large	2	10
is4gen.xlarge	3	20
is4gen.2xlarge	3	40
is4gen.4xlarge	7	60
is4gen.8xlarge	7	60

Accelerated computing

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
dl1.24xlarge	59	120
dl2q.24xlarge	14	120
f2.6xlarge	7	90
f2.12xlarge	7	120
f2.48xlarge	14	120
g4ad.xlarge	1	12
g4ad.2xlarge	1	12
g4ad.4xlarge	2	12
g4ad.8xlarge	3	12
g4ad.16xlarge	7	12
g5.xlarge	3	6
g5.2xlarge	3	19
g5.4xlarge	7	40
g5.8xlarge	7	90
g5.12xlarge	14	120
g5.16xlarge	7	120
g5.24xlarge	14	120
g5.48xlarge	6	120
g5g.xlarge	3	20

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
g5g.2xlarge	3	40
g5g.4xlarge	7	60
g5g.8xlarge	7	60
g5g.16xlarge	14	120
g5g.metal	14	120
g6.xlarge	3	20
g6.2xlarge	3	40
g6.4xlarge	7	60
g6.8xlarge	7	90
g6.12xlarge	7	120
g6.16xlarge	14	120
g6.24xlarge	14	120
g6.48xlarge	14	120
g6e.xlarge	3	20
g6e.2xlarge	3	40
g6e.4xlarge	7	60
g6e.8xlarge	7	90
g6e.12xlarge	9	120
g6e.16xlarge	14	120
g6e.24xlarge	19	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
g6e.48xlarge	39	120
g6f.large	1	10
g6f.xlarge	3	20
g6f.2xlarge	3	40
g6f.4xlarge	7	60
gr6.4xlarge	7	60
gr6.8xlarge	7	90
gr6f.4xlarge	7	60
inf2.xlarge	3	20
inf2.8xlarge	7	90
inf2.24xlarge	14	120
inf2.48xlarge	14	120
p4d.24xlarge	59	120
p4de.24xlarge	59	120
p5.4xlarge	3	60
p5.48xlarge	63	242
p5e.48xlarge	63	242
p5en.48xlarge	63	242
p6-b200.48xlarge	31	242
p6e-gb200.36xlarge	38	120

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
trn1.2xlarge	3	19
trn1.32xlarge	39	120
trn1n.32xlarge	79	242
trn2.48xlarge	31	242
trn2u.48xlarge	31	242
vt1.3xlarge	3	40
vt1.6xlarge	7	60
vt1.24xlarge	14	120

High performance computing

Instance type	Task limit without ENI trunking	Task limit with ENI trunking
hpc6a.48xlarge	1	120
hpc6id.32xlarge	1	120
hpc7g.4xlarge	3	120
hpc7g.8xlarge	3	120
hpc7g.16xlarge	3	120

Reserving Amazon ECS Linux container instance memory

When the Amazon ECS container agent registers a container instance to a cluster, the agent must determine how much memory the container instance has available to reserve for your tasks. Because of platform memory overhead and memory occupied by the system kernel, this number is different than the installed memory amount that is advertised for Amazon EC2 instances. For example, an m4.large instance has 8 GiB of installed memory. However, this does not always translate to exactly 8192 MiB of memory available for tasks when the container instance registers.

The Amazon ECS container agent provides a configuration variable called ECS_RESERVED_MEMORY, which you can use to remove a specified number of MiB of memory from the pool that is allocated to your tasks. This effectively reserves that memory for critical system processes.

If you occupy all of the memory on a container instance with your tasks, then it is possible that your tasks will contend with critical system processes for memory and possibly start a system failure.

For example, if you specify ECS_RESERVED_MEMORY=256 in your container agent configuration file, then the agent registers the total memory minus 256 MiB for that instance, and 256 MiB of memory could not be allocated by ECS tasks. For more information about agent configuration variables and how to set them, see Amazon ECS container agent configuration and Bootstrapping Amazon ECS Linux container instances to pass data.

If you specify 8192 MiB for the task, and none of your container instances have 8192 MiB or greater of memory available to satisfy this requirement, then the task cannot be placed in your cluster. If you are using a managed compute environment, then AWS Batch must launch a larger instance type to accommodate the request.

You should also reserve some memory for the Amazon ECS container agent and other critical system processes on your container instances, so that your task's containers do not contend for the same memory and possibly starts a system failure.

The Amazon ECS container agent uses the Docker ReadMemInfo() function to query the total memory available to the operating system. Both Linux and Windows provide command line utilities to determine the total memory.

Example - Determine Linux total memory

The **free** command returns the total memory that is recognized by the operating system.

```
$ free -b
```

Example output for an m4.large instance running the Amazon ECS-optimized Amazon Linux AMI.

```
             total       used       free     shared    buffers     cached
Mem:    8373026816  348180480 8024846336      90112   25534464  205418496
```

```
-/+ buffers/cache:  117227520 8255799296
```

This instance has 8373026816 bytes of total memory, which translates to 7985 MiB available for tasks.

Example - Determine Windows total memory

The **wmic** command returns the total memory that is recognized by the operating system.

```
C:\> wmic ComputerSystem get TotalPhysicalMemory
```

Example output for an `m4.large` instance running the Amazon ECS-optimized Windows Server AMI.

```
TotalPhysicalMemory
8589524992
```

This instance has 8589524992 bytes of total memory, which translates to 8191 MiB available for tasks.

Viewing container instance memory

You can view how much memory a container instance registers with in the Amazon ECS console (or with the DescribeContainerInstances API operation). If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, you can observe the memory available for that container instance and then assign your tasks that much memory.

To view container instance memory

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**, and then choose the cluster that hosts your container instance.
3. Choose **Infrastructure**, and then under Container instances, choose a container instance.
4. The **Resources** section shows the registered and available memory for the container instance.

   The **Registered** memory value is what the container instance; registered with Amazon ECS when it was first launched, and the **Available** memory value is what has not already been allocated to tasks.

Managing Amazon ECS container instances remotely using AWS Systems Manager

You can use the Run Command capability in AWS Systems Manager (Systems Manager) to securely and remotely manage the configuration of your Amazon ECS container instances. Run Command provides a simple way to perform common administrative tasks without logging on locally to the instance. You can manage configuration changes across your clusters by simultaneously executing commands on multiple container instances. Run Command reports the status and results of each command.

Here are some examples of the types of tasks you can perform with Run Command:

- Install or uninstall packages.
- Perform security updates.
- Clean up Docker images.
- Stop or start services.
- View system resources.
- View log files.
- Perform file operations.

For more information about Run Command, see AWS Systems Manager Run Command in the *AWS Systems Manager User Guide*.

The following are prequisites to using Systems Manager with Amazon ECS.

1. You must grant the container instance role (**ecsInstanceRole**) permissions to access the Systems Manager APIs. You can do this by assigning the **AmazonSSMManagedInstanceCore** to the `ecsInstanceRole` role. For information about how to attach a policy to a role, see Update permissions for a role in the *AWS Identity and Access Management User Guide*
2. Verify that SSM Agent is installed on your container instances. For more information, see Manually installing and uninstalling SSM Agent on EC2 instances for Linux.

After you attach Systems Manager managed policies to your `ecsInstanceRole` and verify that AWS Systems Manager Agent (SSM Agent) is installed on your container instances, you can start using Run Command to send commands to your container instances. For information about running commands and shell scripts on your instances and viewing the resulting output, see

Running Commands Using Systems Manager Run Command and Run Command Walkthroughs in the *AWS Systems Manager User Guide*.

A common use case is to update container instance software with Run Command. You can follow the procedues in the AWS Systems Manager User Guide with the following parameters.

Parameter	Value	
Command document	`AWS-RunShellScript`	
Command	`$ yum update -y`	
Target instances	Your container instances	

Using an HTTP proxy for Amazon ECS Linux container instances

You can configure your Amazon ECS container instances to use an HTTP proxy for both the Amazon ECS container agent and the Docker daemon. This is useful if your container instances do not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

To configure your Amazon ECS Linux container instance to use an HTTP proxy, set the following variables in the relevant files at launch time (with Amazon EC2 user data). You can also manually edit the configuration file, and then restart the agent.

`/etc/ecs/ecs.config` (Amazon Linux 2 and AmazonLinux AMI)

`HTTP_PROXY=`*`10.0.0.131`*`:`*`3128`*

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Amazon ECS agent to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

`NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock`

Set this value to `169.254.169.254,169.254.170.2,/var/run/docker.sock` to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

`/etc/systemd/system/ecs.service.d/http-proxy.conf` (Amazon Linux 2 only)

`Environment="HTTP_PROXY=10.0.0.131:3128/"`

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for `ecs-init` to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

`Environment="NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock"`

Set this value to `169.254.169.254,169.254.170.2,/var/run/docker.sock` to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

`/etc/init/ecs.override` (Amazon Linux AMI only)

`env HTTP_PROXY=10.0.0.131:3128`

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for `ecs-init` to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

`env NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock`

Set this value to `169.254.169.254,169.254.170.2,/var/run/docker.sock` to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

`/etc/systemd/system/docker.service.d/http-proxy.conf` (Amazon Linux 2 only)

`Environment="HTTP_PROXY=http://10.0.0.131:3128"`

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Docker daemon to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

`Environment="NO_PROXY=169.254.169.254,169.254.170.2"`

Set this value to `169.254.169.254,169.254.170.2` to filter EC2 instance metadata from the proxy.

`/etc/sysconfig/docker` (Amazon Linux AMI and Amazon Linux 2 only)

`export HTTP_PROXY=http://10.0.0.131:3128`

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Docker daemon to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

`export NO_PROXY=169.254.169.254,169.254.170.2`

Set this value to `169.254.169.254,169.254.170.2` to filter EC2 instance metadata from the proxy.

Setting these environment variables in the above files only affects the Amazon ECS container agent, `ecs-init`, and the Docker daemon. They do not configure any other services (such as **yum**) to use the proxy.

For information about how to confiure thhe proxy, see [How do I set up an HTTP proxy for Docker and the Amazon ECS container agent in Amazon Linux 2 or AL2023]().

Configuring pre-initialized instances for your Amazon ECS Auto Scaling group

Amazon ECS supports Amazon EC2 Auto Scaling warm pools. A warm pool is a group of pre-initialized Amazon EC2 instances ready to be placed into service. Whenever your application needs to scale out, Amazon EC2 Auto Scaling uses the pre-initialized instances from the warm pool rather than launching cold instances, allows for any final initialization process to run, and then places the instance into service.

To learn more about warm pools and how to add a warm pool to your Auto Scaling group, see [Warm pools for Amazon EC2 Auto Scaling]() *in the Amazon EC2 Auto Scaling User Guide*.

When you create or update a warm pool for an Auto Scaling group for Amazon ECS , you cannot set the option that returns instances to the warm pool on scale in (`ReuseOnScaleIn`). For more information, see [put-warm-pool]() in the *AWS Command Line Interface Reference*.

To use warm pools with your Amazon ECS cluster, set the `ECS_WARM_POOLS_CHECK` agent configuration variable to `true` in the **User data** field of your Amazon EC2 Auto Scaling group launch template.

The following shows an example of how the agent configuration variable can be specified in the **User data** field of an Amazon EC2 launch template. Replace *MyCluster* with the name our your cluster.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_WARM_POOLS_CHECK=true
EOF
```

The `ECS_WARM_POOLS_CHECK` variable is only supported on agent versions `1.59.0` and later. For more information about the variable, see Amazon ECS container agent configuration.

Updating the Amazon ECS container agent

Occasionally, you might need to update the Amazon ECS container agent to pick up bug fixes and new features. Updating the Amazon ECS container agent does not interrupt running tasks or services on the container instance. The process for updating the agent differs depending on whether your container instance was launched with an Amazon ECS-optimized AMI or another operating system.

> **Note**
>
> Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

Checking the Amazon ECS container agent version

You can check the version of the container agent that is running on your container instances to see if you need to update it. The container instance view in the Amazon ECS console provides the agent version. Use the following procedure to check your agent version.

Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the external instance.

4. On the **Cluster : *name*** page, choose the **Infrastructure** tab.
5. Under **Container instances**, note the **Agent version** column for your container instances. If the container instance does not contain the latest version of the container agent, the console alerts you with a message and flags the outdated agent version.

   If your agent version is outdated, you can update your container agent with the following procedures:

   - If your container instance is running an Amazon ECS-optimized AMI, see Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI.
   - If your container instance is not running an Amazon ECS-optimized AMI, see Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs).

> **Important**
>
> To update the Amazon ECS agent version from versions before v1.0.0 on your Amazon ECS-optimized AMI, we recommend that you terminate your current container instance and launch a new instance with the most recent AMI version. Any container instances that use a preview version should be retired and replaced with the most recent AMI. For more information, see Launching an Amazon ECS Linux container instance.

Amazon ECS container agent introspection API

You can also use the to check the agent Amazon ECS container agent introspection API version from the container instance itself. For more information, see Amazon ECS container introspection.

To check if your Amazon ECS container agent is running the latest version with the introspection API

1. Log in to your container instance via SSH.
2. Query the introspection API.

```
[ec2-user ~]$ curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

> **Note**
>
> The introspection API added `Version` information in the version v1.0.0 of the Amazon ECS container agent. If `Version` is not present when querying the introspection API, or the introspection API is not present in your agent at all, then the version you are running is v0.0.3 or earlier. You should update your version.

Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI

If you are using an Amazon ECS-optimized AMI, you have several options to get the latest version of the Amazon ECS container agent (shown in order of recommendation):

- Terminate the container instance and launch the latest version of the Amazon ECS-optimized Amazon Linux 2 AMI (either manually or by updating your Auto Scaling launch configuration with the latest AMI). This provides a fresh container instance with the most current tested and validated versions of Amazon Linux, Docker, `ecs-init`, and the Amazon ECS container agent. For more information, see Amazon ECS-optimized Linux AMIs.
- Connect to the instance with SSH and update the `ecs-init` package (and its dependencies) to the latest version. This operation provides the most current tested and validated versions of Docker and `ecs-init` that are available in the Amazon Linux repositories and the latest version of the Amazon ECS container agent. For more information, see To update the `ecs-init` package on an Amazon ECS-optimized AMI.
- Update the container agent with the `UpdateContainerAgent` API operation, either through the console or with the AWS CLI or AWS SDKs. For more information, see Updating the Amazon ECS container agent with the `UpdateContainerAgent` API operation.

> **Note**
>
> Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

To update the `ecs-init` package on an Amazon ECS-optimized AMI

1. Log in to your container instance via SSH.

2. Update the `ecs-init` package with the following command.

```
sudo yum update -y ecs-init
```

> **Note**
>
> The `ecs-init` package and the Amazon ECS container agent are updated immediately. However, newer versions of Docker are not loaded until the Docker daemon is restarted. Restart either by rebooting the instance, or by running the following commands on your instance:
>
> - Amazon ECS-optimized Amazon Linux 2 AMI:
>
> ```
> sudo systemctl restart docker
> ```
>
> - Amazon ECS-optimized Amazon Linux AMI:
>
> ```
> sudo service docker restart && sudo start ecs
> ```

Updating the Amazon ECS container agent with the `UpdateContainerAgent` API operation

> **Important**
>
> The `UpdateContainerAgent` API is only supported on Linux variants of the Amazon ECS-optimized AMI, with the exception of the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI. For container instances using the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, update the `ecs-init` package to update the agent. For container instances that are running other operating systems, see Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs). If you are using Windows container instances, we recommend that you launch new container instances to update the agent version in your Windows clusters.

The `UpdateContainerAgent` API process begins when you request an agent update, either through the console or with the AWS CLI or AWS SDKs. Amazon ECS checks your current agent version against the latest available agent version, and if an update is possible. If an update is not available, for example, if the agent is already running the most recent version, then a `NoUpdateAvailableException` is returned.

The stages in the update process shown above are as follows:

`PENDING`

An agent update is available, and the update process has started.

`STAGING`

The agent has begun downloading the agent update. If the agent cannot download the update, or if the contents of the update are incorrect or corrupted, then the agent sends a notification of the failure and the update transitions to the `FAILED` state.

`STAGED`

The agent download has completed and the agent contents have been verified.

`UPDATING`

The `ecs-init` service is restarted and it picks up the new agent version. If the agent is for some reason unable to restart, the update transitions to the `FAILED` state; otherwise, the agent signals Amazon ECS that the update is complete.

> **Note**
>
> Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

To update the Amazon ECS container agent on an Amazon ECS-optimized AMI in the console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster.
4. On the **Cluster : *name*** page, choose the **Infrastructure** tab.
5. Under **Container instances**, select the instances to update, and then choose **Actions**, **Update agent**.

Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs)

Occasionally, you might need to update the Amazon ECS container agent to pick up bug fixes and new features. Updating the Amazon ECS container agent does not interrupt running tasks or services on the container instance.

> **Note**
>
> Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

1. Log in to your container instance via SSH.
2. Check to see if your agent uses the `ECS_DATADIR` environment variable to save its state.

   ```
   ubuntu:~$ docker inspect ecs-agent | grep ECS_DATADIR
   ```

   Output:

   ```
   "ECS_DATADIR=/data",
   ```

   > **Important**
   >
   > If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before updating your agent. Newer agents with the `ECS_DATADIR` environment variable save their state and you can update them while tasks are running without issues.

3. Stop the Amazon ECS container agent.

   ```
   ubuntu:~$ docker stop ecs-agent
   ```

4. Delete the agent container.

   ```
   ubuntu:~$ docker rm ecs-agent
   ```

5. Ensure that the `/etc/ecs` directory and the Amazon ECS container agent configuration file exist at `/etc/ecs/ecs.config`.

```
ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
```

6. Edit the `/etc/ecs/ecs.config` file and ensure that it contains at least the following variable declarations. If you do not want your container instance to register with the default cluster, specify your cluster name as the value for `ECS_CLUSTER`.

```
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
```

For more information about these and other agent runtime options, see Amazon ECS container agent configuration.

> **Note**
>
> You can optionally store your agent environment variables in Amazon S3 (which can be downloaded to your container instances at launch time using Amazon EC2 user data). This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see Storing Amazon ECS container instance configuration in Amazon S3 and Using non-AWS container images in Amazon ECS.

7. Pull the latest Amazon ECS container agent image from Amazon Elastic Container Registry Public.

```
ubuntu:~$ docker pull public.ecr.aws/ecs/amazon-ecs-agent:latest
```

Output:

```
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
511136ea3c5a: Download complete
9950b5d678a1: Download complete
c48ddcf21b63: Download complete
Status: Image is up to date for amazon/amazon-ecs-agent:latest
```

8. Run the latest Amazon ECS container agent on your container instance.

> **Note**
>
> Use Docker restart policies or a process manager (such as **upstart** or **systemd**) to treat the container agent as a service or a daemon and ensure that it is restarted after exiting. The Amazon ECS-optimized AMI uses the `ecs-init` RPM for this purpose, and you can view the source code for this RPM on GitHub.

The following example of the agent run command is broken into separate lines to show each option. For more information about these and other agent runtime options, see Amazon ECS container agent configuration.

> **Important**
>
> Operating systems with SELinux enabled require the `--privileged` option in your **docker run** command. In addition, for SELinux-enabled container instances, we recommend that you add the `:Z` option to the `/log` and `/data` volume mounts. However, the host mounts for these volumes must exist before you run the command or you receive a `no such file or directory` error. Take the following action if you experience difficulty running the Amazon ECS agent on an SELinux-enabled container instance:
>
> - Create the host volume mount points on your container instance.
>
> ```
> ubuntu:~$ sudo mkdir -p /var/log/ecs /var/lib/ecs/data
> ```
>
> - Add the `--privileged` option to the **docker run** command below.
> - Append the `:Z` option to the `/log` and `/data` container volume mounts (for example, `--volume=/var/log/ecs/:/log:Z`) to the **docker run** command below.

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run:/var/run \
--volume=/var/log/ecs/:/log \
```

```
--volume=/var/lib/ecs/data:/data \
--volume=/etc/ecs:/etc/ecs \
--volume=/etc/ecs:/etc/ecs/pki \
--net=host \
--env-file=/etc/ecs/ecs.config \
amazon/amazon-ecs-agent:latest
```

> **Note**
>
> If you receive an `Error response from daemon: Cannot start container` message, you can delete the failed container with the **sudo docker rm ecs-agent** command and try running the agent again.

Amazon ECS-optimized Windows AMIs

The Amazon ECS-optimized AMIs are preconfigured with the necessary components that you need to run Amazon ECS workloads. Although you can create your own container instance AMI that meets the basic specifications needed to run your containerized workloads on Amazon ECS, the Amazon ECS-optimized AMIs are preconfigured and tested on Amazon ECS by AWS engineers. It is the simplest way for you to get started and to get your containers running on AWS quickly.

The Amazon ECS-optimized AMI metadata, including the AMI name, Amazon ECS container agent version, and Amazon ECS runtime version which includes the Docker version, for each variant can be retrieved programmatically. For more information, see the section called “Retrieving Amazon ECS-optimized Windows AMI metadata”.

> **Important**
>
> All ECS-optimized AMI variants produced after August 2022 will be migrating from Docker EE (Mirantis) to Docker CE (Moby project).
> To ensure that customers have the latest security updates by default, Amazon ECS maintains at least the last three Windows Amazon ECS-optimized AMIs. After releasing new Windows Amazon ECS-optimized AMIs, Amazon ECS makes the Windows Amazon ECS-optimized AMIs that are older private. If there is a private AMI that you need access to, let us know by filing a ticket with Cloud Support.

Amazon ECS-optimized AMI variants

The following Windows Server variants of the Amazon ECS-optimized AMI are available for your Amazon EC2 instances.

> **Important**
>
> All ECS-optimized AMI variants produced after August will be migrating from Docker EE (Mirantis) to Docker CE (Moby project).

- **Amazon ECS-optimized Windows Server 2025 Full AMI**
- **Amazon ECS-optimized Windows Server 2025 Core AMI**
- **Amazon ECS-optimized Windows Server 2022 Full AMI**
- **Amazon ECS-optimized Windows Server 2022 Core AMI**
- **Amazon ECS-optimized Windows Server 2019 Full AMI**
- **Amazon ECS-optimized Windows Server 2019 Core AMI**
- **Amazon ECS-optimized Windows Server 2016 Full AMI**

> **Important**
>
> Windows Server 2016 does not support the latest Docker version, for example 25.x.x. Therefore the Windows Server 2016 Full AMIs will not receive security or bug patches to the Docker runtime. We recommend that you move to one of the following Windows platforms:
>
> - Windows Server 2022 Full
> - Windows Server 2022 Core
> - Windows Server 2019 Full
> - Windows Server 2019 Core

On August 9, 2022, the Amazon ECS-optimized Windows Server 20H2 Core AMI reached its end of support date. No new versions of this AMI will be released. For more information, see Windows Server release information.

Windows Server 2025, Windows Server 2022, Windows Server 2019, and Windows Server 2016 are Long-Term Servicing Channel (LTSC) releases. Windows Server 20H2 is a Semi-Annual Channel (SAC) release. For more information, see Windows Server release information.

Considerations

Here are some things you should know about Amazon EC2 Windows containers and Amazon ECS.

- Windows containers can't run on Linux container instances, and the opposite is also the case. For better task placement for Windows and Linux tasks, keep Windows and Linux container instances in separate clusters and only place Windows tasks on Windows clusters. You can ensure that Windows task definitions are only placed on Windows instances by setting the following placement constraint: `memberOf(ecs.os-type=='windows')`.
- Windows containers are supported for tasks that use the EC2 and Fargate launch types.
- Windows containers and container instances can't support all the task definition parameters that are available for Linux containers and container instances. For some parameters, they aren't supported at all, and others behave differently on Windows than they do on Linux. For more information, see Amazon ECS task definition differences for EC2 instances running Windows.
- For the IAM roles for tasks feature, you need to configure your Windows container instances to allow the feature at launch. Your containers must run some provided PowerShell code when they use the feature. For more information, see Amazon EC2 Windows instance additional configuration.
- The IAM roles for tasks feature uses a credential proxy to provide credentials to the containers. This credential proxy occupies port 80 on the container instance, so if you use IAM roles for tasks, port 80 is not available for tasks. For web service containers, you can use an Application Load Balancer and dynamic port mapping to provide standard HTTP port 80 connections to your containers. For more information, see Use load balancing to distribute Amazon ECS service traffic.
- The Windows Server Docker images are large (9 GiB). So, your Windows container instances require more storage space than Linux container instances.
- To run a Windows container on a Windows Server, the container's base image OS version must match that of the host. For more information, see Windows container version compatibility on the Microsoft documentation website. If your cluster runs multiple Windows versions, you can ensure that a task is placed on an EC2 instance running on the same version by using the placement constraint: `memberOf(attribute:ecs.os-family ==`

`WINDOWS_SERVER_<OS_Release>_<FULL or CORE>`). For more information, see [the section called “Retrieving Amazon ECS-optimized Windows AMI metadata”](#).

Retrieving Amazon ECS-optimized Windows AMI metadata

The AMI ID, image name, operating system, container agent version, and runtime version for each variant of the Amazon ECS-optimized AMIs can be programmatically retrieved by querying the Systems Manager Parameter Store API. For more information about the Systems Manager Parameter Store API, see [GetParameters](#) and [GetParametersByPath](#).

> **Note**
>
> Your administrative user must have the following IAM permissions to retrieve the Amazon ECS-optimized AMI metadata. These permissions have been added to the `AmazonECS_FullAccess` IAM policy.
>
> - ssm:GetParameters
> - ssm:GetParameter
> - ssm:GetParametersByPath

Systems Manager Parameter Store parameter format

> **Note**
>
> The following Systems Manager Parameter Store API parameters are deprecated and should not be used to retrieve the latest Windows AMIs:
>
> - `/aws/service/ecs/optimized-ami/windows_server/2016/english/full/recommended/image_id`
> - `/aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended/image_id`

The following is the format of the parameter name for each Amazon ECS-optimized AMI variant.

- Windows Server 2025 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized
```

- Windows Server 2025 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized
```

- Windows Server 2022 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

- Windows Server 2022 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

- Windows Server 2019 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

- Windows Server 2019 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

- Windows Server 2016 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

The following parameter name format retrieves the metadata of the latest stable Windows Server 2019 Full AMI

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-
English-Full-ECS_Optimized
```

The following is an example of the JSON object that is returned for the parameter value.

```
{
    "Parameters": [
        {
            "Name": "/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-
ECS_Optimized",
```

```
            "Type": "String",
            "Value": "{\"image_name\":\"Windows_Server-2019-English-Full-ECS_Optimized-2023.06.13\",\"image_id\":\"ami-0debc1fb48e4aee16\",\"ecs_runtime_version\":\"Docker (CE) version 20.10.21\",\"ecs_agent_version\":\"1.72.0\"}",
            "Version": 58,
            "LastModifiedDate": "2023-06-22T19:37:37.841000-04:00",
            "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized",
            "DataType": "text"
        }
    ],
    "InvalidParameters": []
}
```

Each of the fields in the output above are available to be queried as sub-parameters. Construct the parameter path for a sub-parameter by appending the sub-parameter name to the path for the selected AMI. The following sub-parameters are available:

- `schema_version`
- `image_id`
- `image_name`
- `os`
- `ecs_agent_version`
- `ecs_runtime_version`

Examples

The following examples show ways in which you can retrieve the metadata for each Amazon ECS-optimized AMI variant.

Retrieving the metadata of the latest stable Amazon ECS-optimized AMI

You can retrieve the latest stable Amazon ECS-optimized AMI using the AWS CLI with the following AWS CLI commands.

- **For the Amazon ECS-optimized Windows Server 2025 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2025 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2022 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2022 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2019 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2019 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2016 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized --region us-east-1
```

Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template

You can reference the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template by referencing the Systems Manager parameter store name.

```
Parameters:
  LatestECSOptimizedAMI:
    Description: AMI ID
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
```

```
    Default: /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-
ECS_Optimized/image_id
```

Amazon ECS-optimized Windows AMI versions

View the current and previous versions of the Amazon ECS-optimized AMIs and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package.

The Amazon ECS-optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see the section called "Retrieving Amazon ECS-optimized Windows AMI metadata".

The following tabs display a list of Windows Amazon ECS-optimized AMIs versions. For details on referencing the Systems Manager Parameter Store parameter in an AWS CloudFormation template, see Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template.

> **Important**
>
> To ensure that customers have the latest security updates by default, Amazon ECS maintains at least the last three Windows Amazon ECS-optimized AMIs. After releasing new Windows Amazon ECS-optimized AMIs, Amazon ECS makes the Windows Amazon ECS-optimized AMIs that are older private. If there is a private AMI that you need access to, let us know by filing a ticket with Cloud Support.
> Windows Server 2016 does not support the latest Docker version, for example 25.x.x. Therefore the Windows Server 2016 Full AMIs will not receive security or bug patches to the Docker runtime. We recommend that you move to one of the following Windows platforms:
>
> - Windows Server 2022 Full
> - Windows Server 2022 Core
> - Windows Server 2019 Full
> - Windows Server 2019 Core

> **Note**
>
> gMSA plugin logging has been migrated from file-based logging (`C:\ProgramData\Amazon\gmsa`) to Windows Event logging with the August 2025 AMI release. The public

log collector script will collect all gMSA logs. For more information, see Collecting container logs with Amazon ECS logs collector .

Windows Server 2025 Full AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2025 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2025 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2025-English-Full-ECS_Optimized-2025.08.24	`1.98.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS_Optimized-2025.08.16	`1.97.1`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS_Optimized-2025.07.16	`1.96.0`	`25.0.6 (Docker CE)`	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2025 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Full-ECS_Optimized
```

Windows Server 2025 Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2025 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2025 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2025-English-Core-ECS_Optimized-2025.08.24	`1.98.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS_Optimized-2025.08.16	`1.97.1`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Core-ECS_Optimized-2025.07.16	`1.96.0`	`25.0.6 (Docker CE)`	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2025 Core AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2025-English-Core-ECS_Optimized
```

Windows Server 2022 Full AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2022 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2022 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Full-ECS_Optimized-2025.08.24	`1.98.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS_Optimized-2025.08.16	`1.97.1`	`25.0.6 (Docker CE)`	Public
Windows_Server-2022-English-Full-ECS_Optimized-2025.07.16	`1.95.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2022-English-Full-ECS_Optimized-2025.06.13	`1.94.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2022-English-Full-ECS_Optimized-2025.05.17	`1.93.1`	`25.0.6 (Docker CE)`	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2022 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

Windows Server 2022 Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2022 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2022 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Core-ECS_Optimized-2025.08.24	`1.98.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS_Optimized-2025.08.16	`1.97.1`	`25.0.6 (Docker CE)`	Public
Windows_Server-2022-English-Core-ECS_Optimized-2025.07.16	`1.95.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2022-English-Core-ECS_Optimized-2025.06.13	`1.94.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2022-English-Core-ECS_Optimized-2025.05.17	`1.93.1`	`25.0.6 (Docker CE)`	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2022 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

Windows Server 2019 Full AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2019 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-English-Full-ECS_Optimized-2025.08.24	`1.98.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS_Optimized-2025.08.16	`1.97.1`	`25.0.6 (Docker CE)`	Public
Windows_Server-2019-English-Full-ECS_Optimized-2025.07.16	`1.95.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2019-English-Full-ECS_Optimized-2025.06.13	`1.94.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2019-English-Full-ECS_Optimized-2025.05.17	`1.93.1`	`25.0.6 (Docker CE)`	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2019 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

Windows Server 2019 Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2019 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2019 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-English-Core-ECS_Optimized-2025.08.24	`1.98.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2025-English-Full-ECS_Optimized-2025.08.16	`1.97.1`	`25.0.6 (Docker CE)`	Public
Windows_Server-2019-English-Core-ECS_Optimized-2025.07.16	`1.95.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2019-English-Core-ECS_Optimized-2025.06.13	`1.94.0`	`25.0.6 (Docker CE)`	Public
Windows_Server-2019-English-	`1.93.1`	`25.0.6 (Docker CE)`	Public

Amazon ECS-optimized Windows Server 2019 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Core-ECS_Optimized-2025.05.17			

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2019 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

Windows Server 2016 Full AMI versions

> **Important**
>
> Windows Server 2016 does not support the latest Docker version, for example 25.x.x. Therefore the Windows Server 2016 Full AMIs will not receive security or bug patches to the Docker runtime. We recommend that you move to one of the following Windows platforms:
>
> - Windows Server 2022 Full
> - Windows Server 2022 Core
> - Windows Server 2019 Full
> - Windows Server 2019 Core

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2016 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016-English-Full-ECS_Optimized-2025.08.16	`1.97.1`	`20.10.23 (Docker CE)`	Public
Windows_Server-2016-English-Full-ECS_Optimized-2025.07.16	`1.95.0`	`20.10.23 (Docker CE)`	Public
Windows_Server-2016-English-Full-ECS_Optimized-2025.06.13	`1.94.0`	`20.10.23 (Docker CE)`	Public
Windows_Server-2016-English-Full-ECS_Optimized-2025.05.17	`1.93.1`	`20.10.23 (Docker CE)`	Public
Windows_Server-2016-English-Full-ECS_Optimized-2025.4.14	`1.91.2`	`20.10.23 (Docker CE)`	Public

Use the following AWS CLI Amazon ECS-optimized Windows Server 2016 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

Building your own Amazon ECS-optimized Windows AMI

Use EC2 Image Builder to build your own custom Amazon ECS-optimized Windows AMI. This makes it easy to use a Windows AMI with your own license on Amazon ECS. Amazon ECS provides a managed Image Builder component which provides the system configuration needed to run Windows instances to host your containers. Each Amazon ECS managed component includes a specific container agent and Docker version. You can customize your image to use either the latest Amazon ECS managed component, or if an older container agent or Docker version is needed you can specify a different component.

For a full walkthrough of using EC2 Image Builder, see [Getting started with EC2 Image Builder] in the *EC2 Image Builder User Guide*.

When building your own Amazon ECS-optimized Windows AMI using EC2 Image Builder, you create an image recipe. Your image recipe must meet the following requirements:

- The **Source image** should be based on Windows Server 2019 Core, Windows Server 2019 Full, Windows Server 2022 Core, or Windows Server 2022 Full. Any other Windows operating system is not supported and may not be compatible with the component.
- When specifying the **Build components**, the `ecs-optimized-ami-windows` component is required. The `update-windows` component is recommended, which ensures the image contains the latest security updates.

  To specify a different component version, expand the **Versioning options** menu and specify the component version you want to use. For more information, see [Listing the `ecs-optimized-ami-windows` component versions].

Listing the `ecs-optimized-ami-windows` component versions

When creating an EC2 Image Builder recipe and specifying the `ecs-optimized-ami-windows` component, you can either use the default option or you can specify a specific component version. To determine what component versions are available, along with the Amazon ECS container agent and Docker versions contained within the component, you can use the AWS Management Console.

To list the available `ecs-optimized-ami-windows` component versions

1. Open the EC2 Image Builder console at https://console.aws.amazon.com/imagebuilder/.
2. On the navigation bar, select the Region that are building your image in.
3. In the navigation pane, under the **Saved configurations** menu, choose **Components**.

4. On the **Components** page, in the search bar type `ecs-optimized-ami-windows` and pull down the qualification menu and select **Quick start (Amazon-managed)**.
5. Use the **Description** column to determine the component version with the Amazon ECS container agent and Docker version your image requires.

Amazon ECS Windows container instance management

When you use EC2 instances for your Amazon ECS workloads, you are responsible for maintaining the instances.

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

Management procedures

- Launching an Amazon ECS Windows container instance
- Bootstrapping Amazon ECS Windows container instances to pass data
- Using an HTTP proxy for Amazon ECS Windows container instances
- Configuring Amazon ECS Windows container instances to receive Spot Instance notices

Launching an Amazon ECS Windows container instance

Your Amazon ECS container instances are created using the Amazon EC2 console. Before you begin, be sure that you've completed the steps in Set up to use Amazon ECS.

For more information about the launch wizard, see Launch an instance using the new launch instance wizard in the *Amazon EC2 User Guide*.

You can use the new Amazon EC2 wizard to launch an instance. You can use the following list for the parameters and leave the parameters not listed as the default. The following instructions take you through each parameter group.

Procedure

Before you begin, complete the steps in Set up to use Amazon ECS.

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation bar at the top of the screen, the current AWS Region is displayed (for example, US East (Ohio)). Select a Region in which to launch the instance. This choice is

important because some Amazon EC2 resources can be shared between Regions, while others can't.

3. From the Amazon EC2 console dashboard, choose **Launch instance**.

Name and tags

The instance name is a tag, where the key is **Name**, and the value is the name that you specify. You can tag the instance, the volumes, and elastic graphics. For Spot Instances, you can tag the Spot Instance request only.

Specifying an instance name and additional tags is optional.

- For **Name**, enter a descriptive name for the instance. If you don't specify a name, the instance can be identified by its ID, which is automatically generated when you launch the instance.
- To add additional tags, choose **Add additional tags**. Choose **Add tag**, and then enter a key and value, and select the resource type to tag. Choose **Add tag** again for each additional tag to add.

Application and OS Images (Amazon Machine Image)

An Amazon Machine Image (AMI) contains the information required to create an instance. For example, an AMI might contain the software that's required to act as a web server, such as Apache, and your website.

For the latest Amazon ECS-optimized AMIs and their values, see Windows Amazon ECS-optimized AMI.

Use the **Search** bar to find a suitable Amazon ECS-optimized AMI published by AWS.

1. Based on your requirements, enter one of the following AMIs in the **Search** bar and press **Enter**.
   - Windows_Server-2022-English-Full-ECS_Optimized
   - Windows_Server-2022-English-Core-ECS_Optimized
   - Windows_Server-2019-English-Full-ECS_Optimized
   - Windows_Server-2019-English-Core-ECS_Optimized
   - Windows_Server-2016-English-Full-ECS_Optimized
2. On the **Choose an Amazon Machine Image (AMI)** page, select the **Community AMIs** tab.
3. From the list that appears, choose a Microsoft-verified AMI with the most recent publish date and click **Select**.

Instance type

The instance type defines the hardware configuration and size of the instance. Larger instance types have more CPU and memory. For more information, see Instance types.

- For **Instance type**, select the instance type for the instance.

  The instance type that you select determines the resources available for your tasks to run on.

Key pair (login)

For **Key pair name**, choose an existing key pair, or choose **Create new key pair** to create a new one.

> **Important**
>
> If you choose the **Proceed without key pair (Not recommended)** option, you won't be able to connect to the instance unless you choose an AMI that is configured to allow users another way to log in.

Network settings

Configure the network settings, as necessary.

- **Networking platform**: Choose **Virtual Private Cloud (VPC)**, and then specify the subnet in the **Network interfaces** section.
- **VPC**: Select an existing VPC in which to create the security group.
- **Subnet**: You can launch an instance in a subnet associated with an Availability Zone, Local Zone, Wavelength Zone, or Outpost.

  To launch the instance in an Availability Zone, select the subnet in which to launch your instance. To create a new subnet, choose **Create new subnet** to go to the Amazon VPC console. When you are done, return to the launch instance wizard and choose the Refresh icon to load your subnet in the list.

  To launch the instance in a Local Zone, select a subnet that you created in the Local Zone.

  To launch an instance in an Outpost, select a subnet in a VPC that you associated with the Outpost.

- **Auto-assign Public IP**: If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

> **Note**
>
> Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.
> For more information about interface VPC endpoints, see Amazon ECS interface VPC endpoints (AWS PrivateLink)
> If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see NAT gateways in the *Amazon VPC User Guide* and Using an HTTP proxy for Amazon ECS Linux container instances in this guide.

- **Firewall (security groups)**: Use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored.
  - To select an existing security group, choose **Select existing security group**, and select the security group that you created in Set up to use Amazon ECS

Configure storage

The AMI you selected includes one or more volumes of storage, including the root volume. You can specify additional volumes to attach to the instance.

You can use the **Simple** view.

- **Storage type**: Configure the storage for your container instance.

  If you are using the Amazon ECS-optimized Amazon Linux AMI, your instance has two volumes configured. The **Root** volume is for the operating system's use, and the second Amazon EBS volume (attached to `/dev/xvdcz`) is for Docker's use.

  You can optionally increase or decrease the volume sizes for your instance to meet your application needs.

Advanced details

For **Advanced details**, expand the section to view the fields and specify any additional parameters for the instance.

- **Purchasing option**: Choose **Request Spot Instances** to request Spot Instances. You also need to set the other fields related to Spot Instances. For more information, see Spot Instance Requests.

  > **Note**
  >
  > If you are using Spot Instances and see a `Not available` message, you may need to choose a different instance type.

  .
- **IAM instance profile**: Select your container instance IAM role. This is usually named `ecsInstanceRole`.

  > **Important**
  >
  > If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see Amazon ECS container instance IAM role.

- (Optional) **User data**: Configure your Amazon ECS container instance with user data, such as the agent environment variables from Amazon ECS container agent configuration. Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:
  - By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing *`your_cluster_name`* with the name of your cluster.

    The `EnableTaskIAMRole` turns on the Task IAM roles feature for the tasks.

    In addition, the following options are available when you use the `awsvpc` network mode.
    - `EnableTaskENI`: This flag turns on task networking and is required when you use the `awsvpc` network mode.

- `AwsvpcBlockIMDS`: This optional flag blocks IMDS access for the task containers running in the `awsvpc` network mode.
- `AwsvpcAdditionalLocalRoutes`: This optional flag allows you to have additional routes in the task namespace.

  Replace `ip-address` with the IP Address for the additional routes, for example 172.31.42.23/32.

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster your_cluster_name -EnableTaskIAMRole -EnableTaskENI -
AwsvpcBlockIMDS -AwsvpcAdditionalLocalRoutes
'["ip-address"]'
</powershell>
```

Bootstrapping Amazon ECS Windows container instances to pass data

When you launch an Amazon EC2 instance, you can pass user data to the EC2 instance. The data can be used to perform common automated configuration tasks and even run scripts when the instance boots. For Amazon ECS, the most common use cases for user data are to pass configuration information to the Docker daemon and the Amazon ECS container agent.

You can pass multiple types of user data to Amazon EC2, including cloud boothooks, shell scripts, and `cloud-init` directives. For more information about these and other format types, see the Cloud-Init documentation.

You can pass this user data when using the Amazon EC2 launch wizard. For more information, see Launching an Amazon ECS Linux container instance.

Default Windows user data

This example user data script shows the default user data that your Windows container instances receive if you use the console. The script below does the following:

- Sets the cluster name to the name you entered.
- Sets the IAM roles for tasks.
- Sets `json-file` and `awslogs` as the available logging drivers.

In addition, the following options are available when you use the `awsvpc` network mode.

- `EnableTaskENI`: This flag turns on task networking and is required when you use the `awsvpc` network mode.
- `AwsvpcBlockIMDS`: This optional flag blocks IMDS access for the task containers running in `awsvpc` network mode.
- `AwsvpcAdditionalLocalRoutes`: This optional flag allows you to have additional routes.

  Replace `ip-address` with the IP Address for the additional routes, for example 172.31.42.23/32.

You can use this script for your own container instances (provided that they are launched from the Amazon ECS-optimized Windows Server AMI).

Replace the `-Cluster` *`cluster-name`* line to specify your own cluster name.

```
<powershell>
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '["json-file","awslogs"]' -EnableTaskENI -AwsvpcBlockIMDS -AwsvpcAdditionalLocalRoutes
'["ip-address"]'
</powershell>
```

For Windows tasks that are configured to use the `awslogs` logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. Use the following syntax.

Replace the `-Cluster` *`cluster-name`* line to specify your own cluster name.

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE",
 $TRUE, "Machine")
Initialize-ECSAgent -Cluster cluster-name -EnableTaskIAMRole -LoggingDrivers '["json-file","awslogs"]'
</powershell>
```

Windows agent installation user data

This example user data script installs the Amazon ECS container agent on an instance launched with a **Windows_Server-2016-English-Full-Containers** AMI. It has been adapted from the agent installation instructions on the [Amazon ECS Container Agent GitHub repository](https://github.com/aws/amazon-ecs-agent) README page.

> **Note**
>
> This script is shared for example purposes. It is much easier to get started with Windows containers by using the Amazon ECS-optimized Windows Server AMI. For more information, see Creating an Amazon ECS cluster for the Fargate launch type.

For information about how to install the Amazon ECS agent on Windows Server 2022 Full, see Issue 3753 on GitHub.

You can use this script for your own container instances (provided that they are launched with a version of the **Windows_Server-2016-English-Full-Containers** AMI). Be sure to replace the *windows* line to specify your own cluster name (if you are not using a cluster called `windows`).

```
<powershell>
# Set up directories the agent uses
New-Item -Type directory -Path ${env:ProgramFiles}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS\data -Force
# Set up configuration
$ecsExeDir = "${env:ProgramFiles}\Amazon\ECS"
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", "windows", "Machine")
[Environment]::SetEnvironmentVariable("ECS_LOGFILE", "${env:ProgramData}\Amazon\ECS\log
\ecs-agent.log", "Machine")
[Environment]::SetEnvironmentVariable("ECS_DATADIR", "${env:ProgramData}\Amazon\ECS
\data", "Machine")
# Download the agent
$agentVersion = "latest"
$agentZipUri = "https://s3.amazonaws.com/amazon-ecs-agent/ecs-agent-windows-
$agentVersion.zip"
$zipFile = "${env:TEMP}\ecs-agent.zip"
Invoke-RestMethod -OutFile $zipFile -Uri $agentZipUri
# Put the executables in the executable directory.
Expand-Archive -Path $zipFile -DestinationPath $ecsExeDir -Force
Set-Location ${ecsExeDir}
# Set $EnableTaskIAMRoles to $true to enable task IAM roles
# Note that enabling IAM roles will make port 80 unavailable for tasks.
[bool]$EnableTaskIAMRoles = $false
if (${EnableTaskIAMRoles}) {
  $HostSetupScript = Invoke-WebRequest https://raw.githubusercontent.com/aws/amazon-
ecs-agent/master/misc/windows-deploy/hostsetup.ps1
  Invoke-Expression $($HostSetupScript.Content)
```

```
}
# Install the agent service
New-Service -Name "AmazonECS" `
        -BinaryPathName "$ecsExeDir\amazon-ecs-agent.exe -windows-service" `
        -DisplayName "Amazon ECS" `
        -Description "Amazon ECS service runs the Amazon ECS agent" `
        -DependsOn Docker `
        -StartupType Manual
sc.exe failure AmazonECS reset=300 actions=restart/5000/restart/30000/restart/60000
sc.exe failureflag AmazonECS 1
Start-Service AmazonECS
</powershell>
```

Using an HTTP proxy for Amazon ECS Windows container instances

You can configure your Amazon ECS container instances to use an HTTP proxy for both the Amazon ECS container agent and the Docker daemon. This is useful if your container instances do not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

To configure your Amazon ECS Windows container instance to use an HTTP proxy, set the following variables at launch time (with Amazon EC2 user data).

`[Environment]::SetEnvironmentVariable("HTTP_PROXY", "http://proxy.mydomain:port", "Machine")`

Set `HTTP_PROXY` to the hostname (or IP address) and port number of an HTTP proxy to use for the Amazon ECS agent to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

`[Environment]::SetEnvironmentVariable("NO_PROXY", "169.254.169.254,169.254.170.2,\\.\pipe\docker_engine", "Machine")`

Set `NO_PROXY` to `169.254.169.254,169.254.170.2,\\.\pipe\docker_engine` to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

Example Windows HTTP proxy user data script

The example user data PowerShell script below configures the Amazon ECS container agent and the Docker daemon to use an HTTP proxy that you specify. You can also specify a cluster into which the container instance registers itself.

To use this script when you launch a container instance, follow the steps in the section called "Launching a container instance". Just copy and paste the PowerShell script below into the **User data** field (be sure to substitute the red example values with your own proxy and cluster information).

> **Note**
>
> The `-EnableTaskIAMRole` option is required to enable IAM roles for tasks. For more information, see Amazon EC2 Windows instance additional configuration.

```
<powershell>
Import-Module ECSTools

$proxy = "http://proxy.mydomain:port"
[Environment]::SetEnvironmentVariable("HTTP_PROXY", $proxy, "Machine")
[Environment]::SetEnvironmentVariable("NO_PROXY", "169.254.169.254,169.254.170.2,\\.
\pipe\docker_engine", "Machine")

Restart-Service Docker
Initialize-ECSAgent -Cluster MyCluster -EnableTaskIAMRole
</powershell>
```

Configuring Amazon ECS Windows container instances to receive Spot Instance notices

Amazon EC2 terminates, stops, or hibernates your Spot Instance when the Spot price exceeds the maximum price for your request or capacity is no longer available. Amazon EC2 provides a Spot Instance interruption notice, which gives the instance a two-minute warning before it is interrupted. If Amazon ECS Spot Instance draining is enabled on the instance, ECS receives the Spot Instance interruption notice and places the instance in `DRAINING` status.

> **Important**
>
> Amazon ECS monitors for the Spot Instance interruption notices that have the `terminate` and `stop` instance-actions. If you specified either the `hibernate` instance interruption behavior when requesting your Spot Instances or Spot Fleet, then Amazon ECS Spot Instance draining is not supported for those instances.

When a container instance is set to DRAINING, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Service tasks on the draining container instance that are in the PENDING state are stopped immediately. If there are container instances in the cluster that are available, replacement service tasks are started on them.

You can turn on Spot Instance draining when you launch an instance. You must set the ECS_ENABLE_SPOT_INSTANCE_DRAINING parameter before you start the container agent. Replace *my-cluster* with the name of your cluster.

```
[Environment]::SetEnvironmentVariable("ECS_ENABLE_SPOT_INSTANCE_DRAINING", "true",
 "Machine")

# Initialize the agent
Initialize-ECSAgent -Cluster my-cluster
```

For more information, see the section called “Launching a container instance”.

Amazon ECS clusters for the external launch type

Amazon ECS Anywhere provides support for registering an *external instance* such as an on-premises server or virtual machine (VM), to your Amazon ECS cluster. External instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, the lack of Elastic Load Balancing support makes running these workloads less efficient. Amazon ECS added a new EXTERNAL launch type that you can use to create services or run tasks on your external instances.

Supported operating systems and system architectures

The following is the list of supported operating systems and system architectures.

- Amazon Linux 2
- Amazon Linux 2023
- CentOS Stream 9
- RHEL 7, RHEL 8, RHEL 9 — Neither Docker or RHEL's open package repositories support installing Docker natively on RHEL. You must ensure that Docker is installed before you run the install script that's described in this document.
- Fedora 32, Fedora 33, Fedora 40
- openSUSE Tumbleweed

- Ubuntu 18, Ubuntu 20, Ubuntu 22, Ubuntu 24
- Debian 10

> **Important**
>
> Debian 9 Long Term Support (LTS support) ended on June 30, 2022 and is no longer supported by Amazon ECS Anywhere.

- Debian 11
- Debian 12
- SUSE Enterprise Server 15
- The `x86_64` and `ARM64` CPU architectures are supported.
- The following Windows operating system versions are supported:
  - Windows Server 2022
  - Windows Server 2019
  - Windows Server 2016
  - Windows Server 20H2

Considerations

Before you start using external instances, be aware of the following considerations.

- You can register an external instance to one cluster at a time. For instructions on how to register an external instance with a different cluster, see Deregistering an Amazon ECS external instance.
- Your external instances require an IAM role that allows them to communicate with AWS APIs. For more information, see Amazon ECS Anywhere IAM role.
- Your external instances should not have a preconfigured instance credential chain defined locally as this will interfere with the registration script.
- To send container logs to CloudWatch Logs, make sure that you create and specify a task execution IAM role in your task definition.
- When an external instance is registered to a cluster, the `ecs.capability.external` attribute is associated with the instance. This attribute identifies the instance as an external instance. You can add custom attributes to your external instances to use as a task placement constraint. For more information, see Custom attributes.

- You can add resource tags to your external instance. For more information, see External container instances.
- ECS Exec is supported on external instances. For more information, see Monitor Amazon ECS containers with ECS Exec.
- The following are additional considerations that are specific to networking with your external instances. For more information, see Networking .
  - Service load balancing isn't supported.
  - Service discovery isn't supported.
  - Tasks that run on external instances must use the `bridge`, `host`, or `none` network modes. The `awsvpc` network mode isn't supported.
  - There are Amazon ECS service domains in each AWS Region. These service domains must be allowed to send traffic to your external instances.
  - The SSM Agent installed on your external instance maintains IAM credentials that are rotated every 30 minutes using a hardware fingerprint. If your external instance loses connection to AWS, the SSM Agent automatically refreshes the credentials after the connection is re-established. For more information, see Validating on-premises servers and virtual machines using a hardware fingerprint in the *AWS Systems Manager User Guide*.
- The `UpdateContainerAgent` API isn't supported. For instructions on how to update the SSM Agent or the Amazon ECS agent on your external instances, see Updating the AWS Systems Manager agent and Amazon ECS container agent on an external instance.
- Amazon ECS capacity providers aren't supported. To create a service or run a standalone task on your external instances, use the `EXTERNAL` launch type.
- SELinux isn't supported.
- Using Amazon EFS volumes or specifying an `EFSVolumeConfiguration` isn't supported.
- Integration with App Mesh isn't supported.
- If you use the console to create an external instance task definition, you must create the task definition with the console JSON editor.
- When you run ECS Anywhere on Windows, you must use your own Windows license on the on-premises infrastructure.
- When you use a non Amazon ECS-optimized AMI, run the following commands on the external container instance to configure rules to use IAM roles for tasks. For more information, see External instance additional configuration.

```
$ sysctl -w net.ipv4.conf.all.route_localnet=1
```

```
$ iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
$ iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --
to-ports 51679
```

Networking

Amazon ECS external instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, such as a web service, the lack of Elastic Load Balancing support makes running these workloads less efficient because there isn't support for placing these workloads behind a load balancer.

The following are additional considerations that are specific to networking with your external instances.

- Service load balancing isn't supported.
- Service discovery isn't supported.
- Linux tasks that run on external instances must use the `bridge`, `host`, or `none` network modes. The `awsvpc` network mode isn't supported.

  For more information about each network mode, see [Amazon ECS task networking options for the EC2 launch type](#).
- Windows tasks that run on external instances must use the `default` network mode.
- There are Amazon ECS service domains in each Region and must be allowed to send traffic to your external instances.
- The SSM Agent installed on your external instance maintains IAM credentials that are rotated every 30 minutes using a hardware fingerprint. If your external instance loses connection to AWS, the SSM Agent automatically refreshes the credentials after the connection is re-established. For more information, see [Validating on-premises servers and virtual machines using a hardware fingerprint](#) in the *AWS Systems Manager User Guide*.

The following domains are used for communication between the Amazon ECS service and the Amazon ECS agent that's installed on your external instance. Make sure that traffic is allowed and that DNS resolution works. For each endpoint, *`region`* represents the Region identifier for an AWS Region that's supported by Amazon ECS, such as `us-east-2` for the US East (Ohio) Region. The endpoints for all Regions that you use should be allowed. For the `ecs-a` and `ecs-t` endpoints, you should include an asterisk (for example, `ecs-a-*`).

- `ecs-a-*.region.amazonaws.com` — This endpoint is used when managing tasks.
- `ecs-t-*.region.amazonaws.com` — This endpoint is used to manage task and container metrics.
- `ecs.region.amazonaws.com` — This is the service endpoint for Amazon ECS.
- `ssm.region.amazonaws.com` — This is the service endpoint for AWS Systems Manager.
- `ec2messages.region.amazonaws.com` — This is the service endpoint that AWS Systems Manager uses to communicate between the Systems Manager agent and the Systems Manager service in the cloud.
- `ssmmessages.region.amazonaws.com` — This is the service endpoint that is required to create and delete session channels with the Session Manager service in the cloud.
- If your tasks require communication with any other AWS services, make sure that those service endpoints are allowed. Example applications include using Amazon ECR to pull container images or using CloudWatch for CloudWatch Logs. For more information, see Service endpoints in the *AWS General Reference*.

Amazon FSx for Windows File Server with ECS Anywhere

In order to use the Amazon FSx for Windows File Server with Amazon ECS external instances you must establish a connection between your on-premises data center and the AWS Cloud. For information about the options for connecting your network to your VPC, see Amazon Virtual Private Cloud Connectivity Options.

gMSA with ECS Anywhere

The following use cases are supported for ECS Anywhere.

- The Active Directory is in the AWS Cloud - For this configuration, you create a connection between your on-premises network and the AWS Cloud using an AWS Direct Connect connection. For information about how to create the connection, see Amazon Virtual Private Cloud Connectivity Options.You create an Active Directory in the AWS Cloud. For information about how to get started with AWS Directory Service, see Setting up AWS Directory Service in the *AWS Directory Service Administration Guide*. You can then join your external instances to the domain using the AWS Direct Connect connection. For information about working with gMSA with Amazon ECS, see the section called "Learn how to use gMSAs for EC2 Windows containers".

- The Active Directory is in the on-premises data center. - For this configuration, you join your external instances to the on-premises Active Directory. You then use the locally available credentials when you run the Amazon ECS tasks.

Creating an Amazon ECS cluster for the External launch type

You create a cluster to define the infrastructure your tasks and services run on.

Before you begin, be sure that you've completed the steps in Set up to use Amazon ECS and assign the appropriate IAM permission. For more information, see the section called “Amazon ECS cluster examples”. The Amazon ECS console provides a simple way to create the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack.

To make the cluster creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

You can modify the following options:

- Add a namespace to the cluster.

  A namespace allows services that you create in the cluster can connect to the other services in the namespace without additional configuration. For more information, see Interconnect Amazon ECS services.
- Configure the cluster for external instances
- Assign a AWS KMS key for your managed storage. For information about how to create a key, see Create a KMS key in the *AWS Key Management Service User Guide*.
- Add tags to help you identify your cluster.

To create a new cluster (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, configure the following:

- For **Cluster name**, enter a unique name.

  The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.
- (Optional) To have the namespace used for Service Connect be different from the cluster name, for **Namespace**, enter a unique name.

6. (Optional) Use Container Insights, expand **Monitoring**, and then choose one of the following options:

   - To use the recommended Container Insights with enhanced observability, choose **Container Insights with enhanced observability**.
   - To use Container Insights, choose **Container Insights**.
7. (Optional) To use ECS Exec to debug tasks in the cluster, expand **Troubleshooting configuration**, and then configure the following:

   - Select **Turn on ECS Exec**.
   - (Optional) For **AWS KMS key for ECS Exec**, enter the ARN of the AWS KMS key you want to use to encrypt the ECS Exec session data.
   - (Optional) For **ECS Exec logging**, choose the log destination:
     - To send logs to CloudWatch Logs, choose **Amazon CloudWatch**.
     - To send logs to Amazon S3, choose **Amazon S3**.
     - To disable logging, choose **None**.
8. (Optional) Encrypt the data on managed storage. Under **Encryption**, for **Managed storage**, enter the ARN of the AWS KMS key you want to use to encrypt the managed storage data.
9. (Optional) To help identify your cluster, expand **Tags**, and then configure your tags.

   [Add a tag] Choose **Add tag** and do the following:

   - For **Key**, enter the key name.
   - For **Value**, enter the key value.
10. Choose **Create**.

Next steps

You must register the instances with the cluster. For more information, see [Registering an external instance to an Amazon ECS cluster](#).

Create a task definition for the external launch type. For more information, see Creating an Amazon ECS task definition using the console

Run your applications as standalone tasks, or as part of a service. For more information, see the following:

- Running an application as an Amazon ECS task
- Creating an Amazon ECS rolling update deployment

Registering an external instance to an Amazon ECS cluster

For each external instance you register with an Amazon ECS cluster, it must have the SSM Agent, the Amazon ECS container agent, and Docker installed. To register the external instance to an Amazon ECS cluster, it must first be registered as an AWS Systems Manager managed instance. You can create the installation script in a few clicks on the Amazon ECS console. The installation script includes an Systems Manager activation key and commands to install each of the required agents and Docker. The installation script must be run on your on-premises server or VM to complete the installation and registration steps.

> **Note**
>
> Before registering your Linux external instance with the cluster, create the `/etc/ecs/ecs.config` file on your external instance and add any container agent configuration parameters that you want. You can't do this after registering the external instance to a cluster. For more information, see Amazon ECS container agent configuration.

AWS Management Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose a cluster to register your external instance to.
5. On the **Cluster : *name*** page, choose the **Infrastructure** tab.
6. On the **Register external instances** page, complete the following steps.

a. For **Activation key duration (in days)**, enter the number of days that the activation key remains active for. After the number of days you entered pass, the key no longer works when registering an external instance.

b. For **Number of instances**, enter the number of external instances that you want to register to your cluster with the activation key.

c. For **Instance role**, choose the IAM role to associate with your external instances. If a role wasn't already created, choose **Create new role** to have Amazon ECS create a role on your behalf. For more information about what IAM permissions are required for your external instances, see Amazon ECS Anywhere IAM role.

d. Copy the registration command. This command should be run on each external instance you want to register to the cluster.

> **Important**
>
> The bash portion of the script must be run as root. If the command isn't run as root, an error is returned.

e. Choose **Close**.

AWS CLI for Linux operating systems

1. Create an Systems Manager activation pair. This is used for Systems Manager managed instance activation. The output includes an `ActivationId` and `ActivationCode`. You use these in a later step. Make sure that you specify the ECS Anywhere IAM role that you created. For more information, see Amazon ECS Anywhere IAM role.

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. On your on-premises server or virtual machine (VM), download the installation script.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh"
```

3. (Optional) On your on-premises server or virtual machine (VM), use the following steps to verify the installation script using the script signature file.

a. Download and install GnuPG. For more information about GNUpg, see the [GnuPG website](). For Linux systems, install gpg using the package manager on your flavor of Linux.

b. Retrieve the Amazon ECS PGP public key.

```
gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
```

c. Download the installation script signature. The signature is an ascii detached PGP signature stored in a file with the `.asc` extension.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh.asc" "https://amazon-
ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh.asc"
```

d. Verify the installation script file using the key.

```
gpg --verify /tmp/ecs-anywhere-install.sh.asc /tmp/ecs-anywhere-install.sh
```

The following is the expected output.

```
gpg: Signature made Tue 25 May 2021 07:16:29 PM UTC
gpg:                using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
 owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. On your on-premises server or virtual machine (VM), run the installation script. Specify the cluster name, Region, and the Systems Manager activation ID and activation code from the first step.

```
sudo bash /tmp/ecs-anywhere-install.sh \
    --region $REGION \
    --cluster $CLUSTER_NAME \
    --activation-id $ACTIVATION_ID \
    --activation-code $ACTIVATION_CODE
```

For an on-premises server or virtual machine (VM) that has the NVIDIA driver installed for GPU workloads, you must add the `--enable-gpu` flag to the installation script. When this flag is specified, the install script verifies that the NVIDIA driver is running and then adds the required configuration variables to run your Amazon ECS tasks. For more information about running GPU workloads and specifying GPU requirements in a task definition, see [Specifying GPUs in an Amazon ECS task definition](#).

```
sudo bash /tmp/ecs-anywhere-install.sh \
    --region $REGION \
    --cluster $CLUSTER_NAME \
    --activation-id $ACTIVATION_ID \
    --activation-code $ACTIVATION_CODE \
    --enable-gpu
```

Use the following steps to register an existing external instance with a different cluster.

To register an existing external instance with a different cluster

1. Stop the Amazon ECS container agent.

```
sudo systemctl stop ecs.service
```

2. Edit the `/etc/ecs/ecs.config` file and on the `ECS_CLUSTER` line, ensure the cluster name matches the name of the cluster to register the external instance with.

3. Remove the existing Amazon ECS agent data.

```
sudo rm /var/lib/ecs/data/agent.db
```

4. Start the Amazon ECS container agent.

```
sudo systemctl start ecs.service
```

AWS CLI for Windows operating systems

1. Create an Systems Manager activation pair. This is used for Systems Manager managed instance activation. The output includes an `ActivationId` and `ActivationCode`. You

use these in a later step. Make sure that you specify the ECS Anywhere IAM role that you created. For more information, see Amazon ECS Anywhere IAM role.

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. On your on-premises server or virtual machine (VM), download the installation script.

```
Invoke-RestMethod -URI "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install.ps1" -OutFile “ecs-anywhere-install.ps1”
```

3. (Optional) The Powershell script is signed by Amazon and therefore, Windows automatically performs the certificate validation on the same. You do not need to perform any manual validation.

   To manually verify the certificate, right-click on the file, navigate to properties and use the Digital Signatures tab to obtain more details.

   This option is only available when the host has the certificate in the certificate store.

   The verification should return information similar to the following:

```
# Verification (PowerShell)
Get-AuthenticodeSignature -FilePath .\ecs-anywhere-install.ps1

SignerCertificate                         Status      Path
-----------------                         ------      ----
EXAMPLECERTIFICATE                        Valid       ecs-anywhere-install.ps1

...

Subject              : CN="Amazon Web Services, Inc.",...

----
```

4. On your on-premises server or virtual machine (VM), run the installation script. Specify the cluster name, Region, and the Systems Manager activation ID and activation code from the first step.

```
.\ecs-anywhere-install.ps1 -Region $Region -Cluster $Cluster -ActivationID $ActivationID -ActivationCode $ActivationCode
```

5. Verify the Amazon ECS container agent is running.

```
Get-Service AmazonECS

Status   Name               DisplayName
------   ----               -----------
Running  AmazonECS          Amazon ECS
```

Use the following steps to register an existing external instance with a different cluster.

To register an existing external instance with a different cluster

1. Stop the Amazon ECS container agent.

   ```
   Stop-Service AmazonECS
   ```

2. Modify the ECS_CLUSTER parameter so that the cluster name matches the name of the cluster to register the external instance with.

   ```
   [Environment]::SetEnvironmentVariable("ECS_CLUSTER", $ECSCluster,
    [System.EnvironmentVariableTarget]::Machine)
   ```

3. Remove the existing Amazon ECS agent data.

   ```
   Remove-Item -Recurse -Force $env:ProgramData\Amazon\ECS\data\*
   ```

4. Start the Amazon ECS container agent.

   ```
   Start-Service AmazonECS
   ```

The AWS CLI can be used to create a Systems Manager activation before running the installation script to complete the external instance registration process.

Deregistering an Amazon ECS external instance

We recommend that you deregister the instance from both Amazon ECS and AWS Systems Manager after you are done with the instance. Following deregistration, the external instance is no longer able to accept new tasks.

If you have tasks that are running on the container instance when you deregister it, the tasks remain running until they stop through some other means. However, these tasks are no longer monitored or accounted for by Amazon ECS. If these tasks on your external instance are part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different instance, if possible.

After you deregister the instance, clean up the remaining AWS resources on the instance. You can then register it to a new cluster.

Procedure

AWS Management Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the external instance.
4. On the **Cluster :** ***name*** page, choose the **Infrastructure** tab.
5. Under **Container instances**, select the external instance ID to deregister. You're redirected to the container instance detail page.
6. On the **Container Instance :** ***id*** page, choose **Deregister**.
7. Review the deregistration message. Select **Deregister from AWS Systems Manager** to also deregister the external instance as an Systems Manager managed instance. Choose **Deregister**.

   Note

   You can deregister the external instance as an Systems Manager managed instance in the Systems Manager console. For instructions, see Deregistering managed nodes in a hybrid and multicloud environment in the *AWS Systems Manager User Guide*.

8. After you deregister the instance, clean up AWS resources on your on-premises server or VM.

Operating system	Steps	

Linux	a. Stop the Amazon ECS container agent and the SSM Agent services on the instance.
	`sudo systemctl stop ecs amazon-ssm-agent`
	b. Remove the Amazon ECS and Systems Manager packages.
	For CentOS 7, CentOS 8, and RHEL 7
	`sudo yum remove -y amazon-ecs-init amazon-ssm-agent`
	For SUSE Enterprise Server 15
	`sudo zypper remove -y amazon-ecs-init amazon-ssm-agent`
	For Debian and Ubuntu
	`sudo apt remove -y amazon-ecs-init amazon-ssm-agent`
	c. Remove the leftover directories.	

Operating system	Steps
	`sudo rm -rf /var/lib/ecs /etc/ecs /var/lib/amazon/ssm /var/log/ecs /var/log/amazon/ssm`

Windows	a. Stop the Amazon ECS container agent and the SSM Agent services on the instance.
	`Stop-Service AmazonECS`
	`Stop-Service AmazonSSMAgent`
	b. Remove the Amazon ECS package.
	`.\ecs-anywhere-install.ps1 -Uninstall`

AWS CLI

1. You need the instance ID and the container instance ARN to deregister the container instance. If you do not have theses values, run the following comands

   Run the following commandto get the instance ID.

   You use the instance ID (`instanceID`) to get the container instance ARN (`containerInstanceARN`).

```
instanceId=$(aws ssm describe-instance-information --region "{{ region }}" |
 jq ".InstanceInformationList[] |select(.IPAddress==\"{{ IPv4 Address }}\")
 | .InstanceId" | tr -d'"'
```

Run the following commands.

You use the `containerInstanceArn` as a parameter in the command to deregister the instance (`deregister-container-instance`).

```
instances=$(aws ecs list-container-instances --cluster "{{ cluster }}" --region
 "{{ region }}" | jq -c '.containerInstanceArns')
containerInstanceArn=$(aws ecs describe-container-instances --cluster
 "{{ cluster }}" --region "{{ region }}" --container-instances $instances
 | jq ".containerInstances[] | select(.ec2InstanceId==\"{{ instanceId }}\")
 | .containerInstanceArn" | tr -d '"')
```

2. Run the following command to drain the instance.

```
aws ecs update-container-instances-state --cluster "{{ cluster }}" --region
 "{{ region }}" --container-instances "{{ containerInstanceArn }}" --status
 DRAINING
```

3. After the container instance finishes draining, run the following command to deregister the instance.

```
aws ecs deregister-container-instance --cluster "{{ cluster }}" --region
 "{{ region }}" --container-instance "{{ containerInstanceArn }}"
```

4. Run the following command to remove the container instance from SSM.

```
aws ssm deregister-managed-instance --region "{{ region }}" --instance-id
 "{{ instanceId }}"
```

5. After you deregister the instance, clean up AWS resources on your on-premises server or VM.

Operating system	Steps	
Linux	a. Stop the Amazon ECS container agent and the	

Operating system	Steps	

	SSM Agent services on the instance.
	`sudo systemctl stop ecs amazon-ssm-agent`
	b. Remove the Amazon ECS and Systems Manager packages.
	`sudo (yum/apt/zypper) remove amazon-ecs-init amazon-ssm-agent`
	c. Remove the leftover directories.
	`sudo rm -rf /var/lib/ecs /etc/ecs /var/lib/amazon/ssm /var/log/ecs /var/log/amazon/ssm`	

Operating system	Steps	

Windows	a. Stop the Amazon ECS container agent and the SSM Agent services on the instance.
	`Stop-Service AmazonECS`
	`Stop-Service AmazonSSMAgent`
	b. Remove the Amazon ECS package.
	`.\ecs-anywhere-install.ps1 -Uninstall`	

Updating the AWS Systems Manager agent and Amazon ECS container agent on an external instance

Your on-premises server or VM must run both the AWS Systems Manager Agent (SSM Agent) and the Amazon ECS container agent when running Amazon ECS workloads. AWS releases new versions of these agents when any capabilities are added or updated. If your external instances are using an earlier version of either agent, you can update them using the following procedures.

Updating the SSM Agent on an external instance

AWS Systems Manager recommends that you automate the process of updating the SSM Agent on your instances. They provide several methods to automate updates. For more information, see Automating updates to SSM Agent in the *AWS Systems Manager User Guide*.

Updating the Amazon ECS agent on an external instance

On your external instances, the Amazon ECS container agent is updated by upgrading the `ecs-init` package. Updating the Amazon ECS agent doesn't interrupt the running tasks or services.

Amazon ECS provides the `ecs-init` package and signature file in an Amazon S3 bucket in each Region. Beginning with `ecs-init` version `1.52.1-1`, Amazon ECS provides separate `ecs-init` packages for use depending on the operating system and system architecture your external instance uses.

Use the following table to determine the `ecs-init` package that you should download based on the operating system and system architecture your external instance uses.

> **Note**
>
> You can determine which operating system and system architecture that your external instance uses by using the following commands.
>
> ```
> cat /etc/os-release
> uname -m
> ```

Operating systems (architecture)	ecs-init package
CentOS 7 (x86_64) CentOS 8 (x86_64) CentOS Stream 9 (x86_64) SUSE Enterprise Server 15 (x86_64) RHEL 7 (x86_64) RHEL 8 (x86_64)	`amazon-ecs-init-latest.x86_64.rpm`
CentOS 7 (aarch64) CentOS 8 (aarch64) CentOS Stream 9 (aarch64) RHEL 7 (aarch64)	`amazon-ecs-init-latest.aarch64.rpm`
Debian 9 (x86_64)	`amazon-ecs-init-latest.amd64.deb`

Operating systems (architecture)	ecs-init package
Debian 10 (x86_64) Debian 11 (x86_64) Debian 12 (x86_64) Ubuntu 18 (x86_64) Ubuntu 20 (x86_64) Ubuntu 22 (x86_64) Ubuntu 24 (x86_64)	
Debian 9 (aarch64) Debian 10 (aarch64) Debian 11 (aarch64) Debian 12 (aarch64) Ubuntu 18 (aarch64) Ubuntu 20 (aarch64) Ubuntu 22 (aarch64) Ubuntu 24 (aarch64)	`amazon-ecs-init-latest.arm64.deb`

Follow these steps to update the Amazon ECS agent.

To update the Amazon ECS agent

1. Confirm the Amazon ECS agent version that you're running.

   ```
   curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
   ```

2. Download the `ecs-init` package for your operating system and system architecture. Amazon ECS provides the `ecs-init` package file in an Amazon S3 bucket in each Region. Make sure

that you replace the *<region>* identifier in the command with the Region name (for example, `us-west-2`) that you're geographically closest to.

amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb
```

3. (Optional) Verify the validity of the `ecs-init` package file using the PGP signature.

   a. Download and install GnuPG. For more information about GNUpg, see the [GnuPG website]. For Linux systems, install `gpg` using the package manager on your flavor of Linux.

   b. Retrieve the Amazon ECS PGP public key.

   ```
   gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
   ```

   c. Download the `ecs-init` package signature. The signature is an ASCII detached PGP signature that's stored in a file with the `.asc` extension. Amazon ECS provides the signature file in an Amazon S3 bucket in each Region. Make sure that you replace the *<region>* identifier in the command with the Region name (for example, `us-west-2`) that you're geographically closest to.

   amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm.asc
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm.asc
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb.asc
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb.asc
```

d. Verify the `ecs-init` package file using the key.

For the `rpm` packages

```
gpg --verify amazon-ecs-init.rpm.asc ./amazon-ecs-init.rpm
```

For the deb packages

```
gpg --verify amazon-ecs-init.deb.asc ./amazon-ecs-init.deb
```

The following is the expected output.

```
gpg: Signature made Fri 14 May 2021 09:31:36 PM UTC
gpg:                using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
     Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. Install the `ecs-init` package.

For the `rpm` package on CentOS 7, CentOS 8, and RHEL 7

```
sudo yum install -y ./amazon-ecs-init.rpm
```

For the `rpm` package on SUSE Enterprise Server 15

```
sudo zypper install -y --allow-unsigned-rpm ./amazon-ecs-init.rpm
```

For the deb package

```
sudo dpkg -i ./amazon-ecs-init.deb
```

5. Restart the `ecs` service.

```
sudo systemctl restart ecs
```

6. Verify the Amazon ECS agent version was updated.

```
curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

Updating an Amazon ECS cluster

You can modify the following cluster properties:

- Set a default capacity provider

  Each cluster can have one or more capacity providers and an optional capacity provider strategy. The capacity provider strategy determines how the tasks are spread across the cluster's capacity providers. When you run a standalone task or create a service, you either use the cluster's default capacity provider strategy or a capacity provider strategy that overrides the default one.

- Turn on Container Insights.

  CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. Container Insights also provides diagnostic information, such as container restart failures, that you use to isolate issues and resolve them quickly. For more information, see the section called “Monitor Amazon ECS containers using Container Insights with enhanced observability”.

- Add tags to help you identify your clusters.

Procedure

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : *name*** page, choose **Update cluster**.
5. To set the default capacity provider, under **Default capacity provider strategy**, choose **Add more**.

   a. For **Capacity provider**, choose the capacity provider.

   b. (Optional) For **Base**, enter the minimum number of tasks that run on the capacity provider.

      You can only set a **Base** value for one capacity provider.

   c. (Optional) For **Weight**, enter the relative percentage of the total number of launched tasks that use the specified capacity provider.

   d. (Optional) Repeat the steps for any additional capacity providers.
6. To turn on or off Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
7. To help identify your cluster, expand **Tags**, and then configure your tags.

   [Add a tag] Choose **Add tag** and do the following:

   - For **Key**, enter the key name.
   - For **Value**, enter the key value.

   [Remove a tag] Choose **Remove** to the right of the tag's Key and Value.
8. Choose **Update**.

Deleting an Amazon ECS cluster

If you are finished using a cluster, you can delete it. After you delete the cluster, it transitions to the `INACTIVE` state. Clusters with an `INACTIVE` status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so you should not rely on `INACTIVE` clusters persisting.

Before you delete a cluster, you must perform the following operations:

- Delete all services in the cluster. For more information, see the section called "Deleting a service".
- Stop all currently running tasks. For more information, see the section called "Stopping a task".
- Deregister all registered container instances in the cluster. For more information, see the section called "Deregistering a container instance".
- Delete the namespace. For more information, see Deleting namespaces in the *AWS Cloud Map Developer Guide*.

Procedure

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select the cluster to delete.
5. In the upper-right of the page, choose **Delete Cluster**.

   A message is displayed when you did not delete all the resources associated with the cluster.
6. In the confirmation box, enter **delete** ***`cluster name`***.

Deregistering an Amazon ECS container instance

> **Important**
>
> This topic is for container instances created in Amazon EC2 only. For more information about deregistering external instances, see Deregistering an Amazon ECS external instance.

When you are finished with an Amazon EC2 backed container instance, you should deregister it from your cluster. Following deregistration, the container instance is no longer able to accept new tasks.

If you have tasks running on the container instance when you deregister it, these tasks remain running until you terminate the instance or the tasks stop through some other means. However, these tasks are orphaned which means they are no longer monitored or accounted for by Amazon ECS. If an orphaned task on your container instance is part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different container instance, if possible. Any containers in orphaned service tasks that are registered with an Application Load Balancer target group are deregistered. They begin connection draining according to the settings on the load balancer or target group. If an orphaned tasks is using the `awsvpc` network mode, their elastic network interfaces are deleted.

If you intend to use the container instance for some other purpose after deregistration, you should stop all of the tasks running on the container instance before deregistration. This stops any orphaned tasks from consuming resources.

When deregistering a container instance, be aware of the following considerations.

- Because each container instance has unique state information, they should not be deregistered from one cluster and re-registered into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances in the new cluster. For more information, see Terminate your instance in the *Amazon EC2 User Guide* and Launching an Amazon ECS Linux container instance.
- If the container instance is managed by an Auto Scaling group or a AWS CloudFormation stack, terminate the instance by updating the Auto Scaling group or AWS CloudFormation stack. Otherwise, the Auto Scaling group or AWS CloudFormation will create a new instance after you terminate it.
- If you terminate a running container instance with a connected Amazon ECS container agent, the agent automatically deregisters the instance from your cluster. Stopped container instances or instances with disconnected agents are not automatically deregistered when terminated.
- Deregistering a container instance removes the instance from a cluster, but it does not terminate the Amazon EC2 instance. If you are finished using the instance, be sure to terminate it to stop billing. For more information, see Terminate your instance in the *Amazon EC2 User Guide*.

Procedure

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, choose the Region where your external instance is registered.

3. In the navigation pane, choose **Clusters** and select the cluster that hosts the instance.
4. On the **Cluster : *name*** page, choose the **Infrastructure** tab.
5. Under **Container instances**, select the instance ID to deregister. You're redirected to the container instance detail page.
6. On the **Container Instance : *id*** page, choose **Deregister**.
7. On the confirmation screen, choose **Deregister**.
8. If you are finished with the container instance, terminate the underlying Amazon EC2 instance. For more information, see Terminate Your Instance in the *Amazon EC2 User Guide*.

Draining Amazon ECS container instances

There might be times when you need to remove a container instance from your cluster, for example, to perform system updates or to scale down the cluster capacity. Amazon ECS provides the ability to transition a container instance to a `DRAINING` status. This is referred to as *container instance draining*. When a container instance is set to `DRAINING`, Amazon ECS prevents new tasks from being scheduled for placement on the container instance.

Draining behavior for services

Any tasks that are part of a service that are in a `PENDING` state are stopped immediately. If there is available container instance capacity in the cluster, the service scheduler will start replacement tasks. If there isn't enough container instance capacity, a service event message will be sent indicating the issue.

Tasks that are part of a service on the container instance that are in a `RUNNING` state are transitioned to a `STOPPED` state. The service scheduler attempts to replace the tasks according to the service's deployment type and deployment configuration parameters, `minimumHealthyPercent` and `maximumPercent`. For more information, see Amazon ECS services and Amazon ECS service definition parameters.

- If `minimumHealthyPercent` is below 100%, the scheduler can ignore `desiredCount` temporarily during task replacement. For example, `desiredCount` is four tasks, a minimum of 50% allows the scheduler to stop two existing tasks before starting two new tasks. If the minimum is 100%, the service scheduler can't remove existing tasks until the replacement tasks are considered healthy. If tasks for services that do not use a load balancer are in the `RUNNING` state, they are considered healthy. Tasks for services that use a load balancer are considered healthy if they are in the RUNNING state and the container instance they are hosted on is reported as healthy by the load balancer.

> **Important**
>
> If you use Spot Instances and minimumHealthyPercent is greater than or equal to 100%, then the service will not have enough time to replace the task before the Spot Instance terminates.

- The maximumPercent parameter represents an upper limit on the number of running tasks during task replacement, which allows you to define the replacement batch size. For example, if desiredCount of four tasks, a maximum of 200% starts four new tasks before stopping the four tasks to be drained (provided that the cluster resources required to do this are available). If the maximum is 100%, then replacement tasks can't start until the draining tasks have stopped.

> **Important**
>
> If both minimumHealthyPercent and maximumPercent are 100%, then the service can't remove existing tasks, and also cannot start replacement tasks. This prevents successful container instance draining and prevents making new deployments.

Draining behavior for standalone tasks

Any standalone tasks in the PENDING or RUNNING state are unaffected; you must wait for them to stop on their own or stop them manually. The container instance will remain in DRAINING status.

A container instance has completed draining when all tasks running on the instance transition to a STOPPED state. The container instance remains in a DRAINING state until it is activated again or deleted. You can verify the state of the tasks on the container instance by using the ListTasks operation with the containerInstance parameter to get a list of tasks on the instance followed by a DescribeTasks operation with the Amazon Resource Name (ARN) or ID of each task to verify the task state.

When you are ready for the container instance to start hosting tasks again, you change the state of the container instance from DRAINING to ACTIVE. The Amazon ECS service scheduler will then consider the container instance for task placement again.

Procedure

The following steps can be used to set a container instance to draining using the new AWS Management Console.

You can also use the UpdateContainerInstancesState API action or the update-container-instances-state command to change the status of a container instance to `DRAINING`.

AWS Management Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose a cluster that hosts your instances.
4. On the **Cluster : *name*** page, choose the **Infrastructure** tab. Then, under **Container instances** select the check box for each container instance you want to drain.
5. Choose **Actions**, **Drain**.

Amazon ECS EC2 Container Instances

The Amazon ECS agent is a process that runs on every container instance that is registered with your cluster. It facilitates the communication between your container instances and Amazon ECS.

> **Note**
>
> On Linux container instances, the agent container mounts top-level directories such as `/lib`, `/lib64`, and `/proc`. This is necessary for ECS features and functionalities such as Amazon EBS volumes, `awsvpc` network mode, Amazon ECS Service Connect, and FireLens for Amazon ECS.

Each Amazon ECS container agent version supports a different feature set and provides bug fixes from previous versions. When possible, we always recommend using the latest version of the Amazon ECS container agent. To update your container agent to the latest version, see Updating the Amazon ECS container agent.

The Amazon ECS container agent contains the `amazon-ecs-pause` image.Amazon ECS uses this image for tasks that use `awsvpc` network mode.

To see which features and enhancements are included with each agent release, see https://github.com/aws/amazon-ecs-agent/releases.

> **Important**
>
> The minimum Docker version for reliable metrics is Docker version `v20.10.13` and newer, which is included in Amazon ECS-optimized AMI `20220607` and newer.
> Amazon ECS agent versions `1.20.0` and newer have deprecated support for Docker versions older than `18.01.0`.

Lifecycle

When the Amazon ECS container agent registers an Amazon EC2 instance to your cluster, the Amazon EC2 instance reports its status as `ACTIVE` and its agent connection status as `TRUE`. This container instance can accept run task requests.

If you stop (not terminate) a container instance, the status remains `ACTIVE`, but the agent connection status transitions to `FALSE` within a few minutes. Any tasks that were running on the container instance stop. If you start the container instance again, the container agent reconnects with the Amazon ECS service, and you are able to run tasks on the instance again.

> **Important**
>
> If you stop and start a container instance, or reboot that instance, some older versions of the Amazon ECS container agent register the instance again without deregistering the original container instance ID. In this case, Amazon ECS lists more container instances in your cluster than you actually have. (If you have duplicate container instance IDs for the same Amazon EC2 instance ID, you can safely deregister the duplicates that are listed as `ACTIVE` with an agent connection status of `FALSE`.) This issue is fixed in the current version of the Amazon ECS container agent. For more information about updating to the current version, see Updating the Amazon ECS container agent.

If you change the status of a container instance to `DRAINING`, new tasks are not placed on the container instance. Any service tasks running on the container instance are removed, if possible, so that you can perform system updates. For more information, see Draining Amazon ECS container instances.

If you deregister or terminate a container instance, the container instance status changes to `INACTIVE` immediately, and the container instance is no longer reported when you list your container instances. However, you can still describe the container instance for one hour following termination. After one hour, the instance description is no longer available.

> **Important**
>
> You can drain the instances manually, or build an Auto Scaling group lifecycle hook to set the instance status to `DRAINING`. See [Amazon EC2 Auto Scaling lifecycle hooks](#) for more information about Auto Scaling lifecycle hooks.

Docker Support

Amazon ECS supports the last two major versions of Docker published on Amazon Linux. Currently, this includes Docker 20.10.x and Docker 25.x.

The minimum required Docker version for Amazon ECS can be found in the Amazon ECS Agent specification file on GitHub.

When using the Amazon ECS-optimized AMI, Docker is pre-installed and configured to work with the Amazon ECS container agent. The AMI includes a Docker version that is tested and supported by Amazon ECS.

> **Note**
>
> While Amazon ECS supports multiple Docker versions, we recommend using the Docker version that comes with the Amazon ECS-optimized AMI for the best compatibility and support.

Amazon ECS-optimized AMI

The Linux variants of the Amazon ECS-optimized AMI use the Amazon Linux 2 AMI as their base. The Amazon Linux 2 source AMI name for each variant can be retrieved by querying the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-optimized Linux AMI metadata. When you launch our container instances from the most recent Amazon ECS-optimized Amazon Linux 2 AMI you receive the current container agent version. To launch a container instance with the latest Amazon ECS-optimized Amazon Linux 2 AMI, see Launching an Amazon ECS Linux container instance.

Additional information

The following pages provide additional information about the changes:

- Amazon ECS Agent changelog on GitHub
- The source code for the `ecs-init` application and the scripts and configuration for packaging the agent are now part of the agent repository. For older versions of `ecs-init` and packaging, see Amazon ecs-init changelog on GitHub
- Amazon Linux 2 release notes.
- Docker Engine release notes in the Docker documentation
- NVIDIA Driver Documentation in the NVIDIA documentation

Amazon ECS container agent configuration

Applies to: EC2 instances

The Amazon ECS container agent supports a number of configuration options, most of which you set through environment variables.

If your container instance was launched with a Linux variant of the Amazon ECS-optimized AMI, you can set these environment variables in the `/etc/ecs/ecs.config` file and then restart the agent. You can also write these configuration variables to your container instances with Amazon EC2 user data at launch time. For more information, see Bootstrapping Amazon ECS Linux container instances to pass data.

If your container instance was launched with a Windows variant of the Amazon ECS-optimized AMI, you can set these environment variables with the PowerShell SetEnvironmentVariable command and then restart the agent. For more information, see Run commands when you launch an EC2 instance with user data input in the *Amazon EC2 User Guide* and the section called “Bootstrapping container instances”.

If you are manually starting the Amazon ECS container agent (for non Amazon ECS-optimized AMIs), you can use these environment variables in the **docker run** command that you use to start the agent. Use these variables with the syntax `--env=`*`VARIABLE_NAME`*`=`*`VARIABLE_VALUE`*. For sensitive information, such as authentication credentials for private repositories, you should store your agent environment variables in a file and pass them all at one time with the `--env-file` *`path_to_env_file`* option. You can use the following commands to add the variables.

```
sudo systemctl stop ecs
sudo vi /etc/ecs/ecs.config
# And add the environment variables with VARIABLE_NAME=VARIABLE_VALUE format.
sudo systemctl start ecs
```

Run the Amazon ECS agent with the host PID namespace

By default, the Amazon ECS agent runs with its own PID namespace. In the following configurations, you can configure the Amazon ECS agent to run with the host PID namespace:

- SELinux enforcing mode is enabled.
- Docker's SELinux security policy is set to true.

You can configure this behavior by setting the `ECS_AGENT_PID_NAMESPACE_HOST` environment variable to `true` in your `/etc/ecs/ecs.config` file. When this variable is enabled, `ecs-init` will start the Amazon ECS agent container with the host's PID namespace (`--pid=host`), allowing the agent to bootstrap itself properly in SELinux-enforcing environments. This feature is available in Amazon ECS agent version `1.94.0` and later.

To enable this feature, add the following line to your `/etc/ecs/ecs.config` file:

```
ECS_AGENT_PID_NAMESPACE_HOST=true
```

After making this change, restart the Amazon ECS agent for the change to take effect:

```
sudo systemctl restart ecs
```

The following features will not work SELinux enforcing mode is enabled and the Docker security policy is set to true, even when `ECS_AGENT_PID_NAMESPACE_HOST=true` is set.

- Amazon ECS Exec
- Amazon EBS task attach
- Service Connect
- FireLens for Amazon ECS

Available parameters

For information about the available Amazon ECS container agent configuration parameters, see [Amazon ECS Container Agent](#) on GitHub.

Storing Amazon ECS container instance configuration in Amazon S3

Amazon ECS container agent configuration is controlled with the environment variable. Linux variants of the Amazon ECS-optimized AMI look for these variables in `/etc/ecs/ecs.config` when the container agent starts and configure the agent accordingly. Non-sensative environment variables, such as `ECS_CLUSTER`, can be passed to the container instance at launch through Amazon EC2 user data and written to this file without consequence. However, other sensitive information, such as your AWS credentials or the `ECS_ENGINE_AUTH_DATA` variable, should never be passed to an instance in user data or written to `/etc/ecs/ecs.config` in a way that would allow them to show up in a `.bash_history` file.

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance configuration at launch. You can store a copy of your `ecs.config` file in a private bucket. You can then use Amazon EC2 user data to install the AWS CLI and copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

To store an `ecs.config` file in Amazon S3

1. You must grant the container instance role (**ecsInstanceRole**) permissions to have read only access to Amazon S3. You can do this by assigning the **AmazonS3ReadOnlyAccess** to the `ecsInstanceRole` role. For information about how to attach a policy to a role, see Update permissions for a role in the *AWS Identity and Access Management User Guide*

2. Create an `ecs.config` file with valid Amazon ECS agent configuration variables using the following format. This example configures private registry authentication. For more information, see Using non-AWS container images in Amazon ECS.

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

> **Note**
>
> For a full list of available Amazon ECS agent configuration variables, see [Amazon ECS Container Agent](#) on GitHub.

3. To store your configuration file, create a private bucket in Amazon S3. For more information, see Creating a bucket in the *Amazon Simple Storage Service User Guide*.
4. Upload the `ecs.config` file to your S3 bucket. For more information, see Uploading objects in the *Amazon Simple Storage Service User Guide*.

To load an `ecs.config` file from Amazon S3 at launch

1. Complete the earlier procedures in this section to allow read-only Amazon S3 access to your container instances and store an `ecs.config` file in a private S3 bucket.
2. Launch new container instances and use the following example script in the EC2 User data. The script installs the AWS CLI and copies your configuration file to `/etc/ecs/ecs.config`. For more information, see Launching an Amazon ECS Linux container instance.

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

Installing the Amazon ECS container agent

If you want to register an Amazon EC2 instance with your Amazon ECS cluster and that instance is not using an AMI based on the Amazon ECS-optimized AMI, you can install the Amazon ECS container agent manually using the following procedure. To do this, you can either download the agent from one of the regional Amazon S3 buckets or from Amazon Elastic Container Registry Public. If you download from one of the regional Amazon S3 buckets, you can optionally verify the validity of the container agent file using the PGP signature.

> **Note**
>
> The `systemd` units for both Amazon ECS and Docker services have a directive to wait for `cloud-init` to finish before starting both services. The `cloud-init` process is not considered finished until your Amazon EC2 user data has finished running. Therefore,

starting Amazon ECS or Docker via Amazon EC2 user data may cause a deadlock. To start the container agent using Amazon EC2 user data you can use `systemctl enable --now --no-block ecs.service`.

Installing the Amazon ECS container agent on a non-Amazon Linux EC2 instance

To install the Amazon ECS container agent on an Amazon EC2 instance, you can download the agent from one of the regional Amazon S3 buckets and install it.

> **Note**
>
> When using a non-Amazon Linux AMI, your Amazon EC2 instance requires `cgroupfs` support for the `cgroup` driver in order for the Amazon ECS agent to support task level resource limits. For more information, see Amazon ECS agent on GitHub.

The latest Amazon ECS container agent files, by Region, for each system architecture are listed below for reference.

Region	Region name	Amazon ECS init deb files	Amazon ECS init rpm files
us-east-2	US East (Ohio)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
us-east-1	US East (N. Virginia)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
us-west-1	US West (N. California)	Amazon ECS init amd64 (amd64)	Amazon ECS init x86_64 (x86_64)

Region	Region name	Amazon ECS init deb files	Amazon ECS init rpm files
		Amazon ECS init arm64 (arm64)	Amazon ECS init aarch64 (aarch64)
us-west-2	US West (Oregon)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-east-1	Asia Pacific (Hong Kong)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-northeast-1	Asia Pacific (Tokyo)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-northeast-2	Asia Pacific (Seoul)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-south-1	Asia Pacific (Mumbai)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)

Region	Region name	Amazon ECS init deb files	Amazon ECS init rpm files
ap-southeast-1	Asia Pacific (Singapore)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ap-southeast-2	Asia Pacific (Sydney)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
ca-central-1	Canada (Central)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
eu-central-1	Europe (Frankfurt)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
eu-west-1	Europe (Ireland)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
eu-west-2	Europe (London)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)

Region	Region name	Amazon ECS init deb files	Amazon ECS init rpm files
eu-west-3	Europe (Paris)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
sa-east-1	South America (São Paulo)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 Amazon ECS init aarch64 (aarch64)
us-gov-east-1	AWS GovCloud (US-East)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)
us-gov-west-1	AWS GovCloud (US-West)	Amazon ECS init amd64 (amd64) Amazon ECS init arm64 (arm64)	Amazon ECS init x86_64 (x86_64) Amazon ECS init aarch64 (aarch64)

To install the Amazon ECS container agent on an Amazon EC2 instance using a non-Amazon Linux AMI

1. Launch an Amazon EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see Amazon ECS container instance IAM role.
2. Connect to your instance.
3. Install the latest version of Docker on your instance.
4. Check your Docker version to verify that your system meets the minimum version requirement.

Note

The minimum Docker version for reliable metrics is Docker version `v20.10.13` and newer, which is included in Amazon ECS-optimized AMI `20220607` and newer. Amazon ECS agent versions `1.20.0` and newer have deprecated support for Docker versions older than `18.01.0`.

```
docker --version
```

5. Download the appropriate Amazon ECS agent file for your operating system and system architecture and install it.

   For `deb` architectures:

   ```
   ubuntu:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.amd64.deb
   ubuntu:~$ sudo dpkg -i amazon-ecs-init-latest.amd64.deb
   ```

   For `rpm` architectures:

   ```
   fedora:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-ecs-agent-us-west-2/amazon-ecs-init-latest.x86_64.rpm
   fedora:~$ sudo yum localinstall -y amazon-ecs-init-latest.x86_64.rpm
   ```

6. Edit the `/lib/systemd/system/ecs.service` file and add the following line at the end of the `[Unit]` section.

   ```
   After=cloud-final.service
   ```

7. (Optional) To register the instance with a cluster other than the `default` cluster, edit the `/etc/ecs/ecs.config` file and add the following contents. The following example specifies the `MyCluster` cluster.

   ```
   ECS_CLUSTER=MyCluster
   ```

   For more information about these and other agent runtime options, see [Amazon ECS container agent configuration](#).

> **Note**
>
> You can optionally store your agent environment variables in Amazon S3 (which can be downloaded to your container instances at launch time using Amazon EC2 user data). This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see Storing Amazon ECS container instance configuration in Amazon S3 and Using non-AWS container images in Amazon ECS.

8. Start the `ecs` service.

```
ubuntu:~$ sudo systemctl start ecs
```

Running the Amazon ECS agent with host network mode

When running the Amazon ECS container agent, `ecs-init` will create the container agent container with the `host` network mode. This is the only supported network mode for the container agent container.

This allows you to block access to the Amazon EC2 instance metadata service endpoint (`http://169.254.169.254`) for the containers started by the container agent. This ensures that containers cannot access IAM role credentials from the container instance profile and enforces that tasks use only the IAM task role credentials. For more information, see Amazon ECS task IAM role.

This also makes it so the container agent doesn't contend for connections and network traffic on the `docker0` bridge.

Amazon ECS container agent log configuration parameters

The Amazon ECS container agent stores logs on your container instances.

For container agent version 1.36.0 and later, by default the logs are located at `/var/log/ecs/ecs-agent.log` on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log` on Windows instances.

For container agent version 1.35.0 and earlier, by default the logs are located at `/var/log/ecs/ecs-agent.log.`*`timestamp`* on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log.`*`timestamp`* on Windows instances.

By default, the agent logs are rotated hourly with a maximum of 24 logs being stored.

The following are the container agent configuration variables that can be used to change the default agent logging behavior. For detailed information about all available configuration parameters, see Amazon ECS container agent configuration or the Amazon ECS Agent README on GitHub.

For container agent version 1.36.0 and later, the following is an example log file when the `logfmt` format is used.

```
level=info time=2019-12-12T23:43:29Z msg="Loading configuration" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-
ecs-agent:latest" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-
ecs-pause:0.1.0" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Amazon ECS agent Version: 1.36.0, Commit:
 ca640387" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Creating root ecs cgroup: /ecs"
 module=init_linux.go
level=info time=2019-12-12T23:43:29Z msg="Creating cgroup /ecs"
 module=cgroup_controller_linux.go
level=info time=2019-12-12T23:43:29Z msg="Loading state!" module=statemanager.go
level=info time=2019-12-12T23:43:29Z msg="Event stream ContainerChange start
 listening..." module=eventstream.go
level=info time=2019-12-12T23:43:29Z msg="Restored cluster 'auto-robc'" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Restored from checkpoint file. I
 am running as 'arn:aws:ecs:us-west-2:0123456789:container-instance/auto-
robc/3330a8a91d15464ea30662d5840164cd' in cluster 'auto-robc'" module=agent.go
```

The following is an example log file when the JSON format is used.

```
{"time": "2019-11-07T22:52:02Z", "level": "info", "msg": "Starting Amazon Elastic
 Container Service Agent", "module": "engine.go"}
```

For container agent versions 1.35.0 and earlier, the following is the format of the log file.

```
2016-08-15T15:54:41Z [INFO] Starting Agent: Amazon ECS Agent - v1.12.0 (895f3c1)
2016-08-15T15:54:41Z [INFO] Loading configuration
2016-08-15T15:54:41Z [WARN] Invalid value for task cleanup duration, will be overridden
 to 3h0m0s, parsed value 0, minimum threshold 1m0s
2016-08-15T15:54:41Z [INFO] Checkpointing is enabled. Attempting to load state
```

```
2016-08-15T15:54:41Z [INFO] Loading state! module="statemanager"
2016-08-15T15:54:41Z [INFO] Detected Docker versions [1.17 1.18 1.19 1.20 1.21 1.22]
2016-08-15T15:54:41Z [INFO] Registering Instance with ECS
2016-08-15T15:54:41Z [INFO] Registered! module="api client"
```

Configuring Amazon ECS container instances for private Docker images

The Amazon ECS container agent can authenticate with private registries, using basic authentication. When you enable private registry authentication, you can use private Docker images in your task definitions. This feature is only supported by tasks using the EC2 launch type.

Another method of enabling private registry authentication uses AWS Secrets Manager to store your private registry credentials securely and then reference them in your container definition. This allows your tasks to use images from private repositories. This method supports tasks using either the EC2 or Fargate launch types. For more information, see [Using non-AWS container images in Amazon ECS](#).

The Amazon ECS container agent looks for two environment variables when it launches:

- `ECS_ENGINE_AUTH_TYPE`, which specifies the type of authentication data that is being sent.
- `ECS_ENGINE_AUTH_DATA`, which contains the actual authentication credentials.

Linux variants of the Amazon ECS-optimized AMI scan the `/etc/ecs/ecs.config` file for these variables when the container instance launches, and each time the service is started (with the **sudo start ecs** command). AMIs that are not Amazon ECS-optimized should store these environment variables in a file and pass them with the `--env-file` *`path_to_env_file`* option to the **docker run** command that starts the container agent.

> **Important**
>
> We do not recommend that you inject these authentication environment variables at instance launch with Amazon EC2 user data or pass them with the `--env` option to the **docker run** command. These methods are not appropriate for sensitive data, such as authentication credentials. For information about safely adding authentication credentials to your container instances, see [Storing Amazon ECS container instance configuration in Amazon S3](#).

Authentication formats

There are two available formats for private registry authentication, `dockercfg` and `docker`.

dockercfg authentication format

The `dockercfg` format uses the authentication information stored in the configuration file that is created when you run the **docker login** command. You can create this file by running **docker login** on your local system and entering your registry user name, password, and email address. You can also log in to a container instance and run the command there. Depending on your Docker version, this file is saved as either `~/.dockercfg` or `~/.docker/config.json`.

```
cat ~/.docker/config.json
```

Output:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "zq212MzEXAMPLE7o6T25Dk0i"
    }
  }
}
```

> **Important**
>
> Newer versions of Docker create a configuration file as shown above with an outer `auths` object. The Amazon ECS agent only supports `dockercfg` authentication data that is in the below format, without the `auths` object. If you have the **jq** utility installed, you can extract this data with the following command: **cat ~/.docker/config.json | jq .auths**

```
cat ~/.docker/config.json | jq .auths
```

Output:

```
{
  "https://index.docker.io/v1/": {
```

```
    "auth": "zq212MzEXAMPLE7o6T25Dk0i",
    "email": "email@example.com"
  }
}
```

In the above example, the following environment variables should be added to the environment variable file (`/etc/ecs/ecs.config` for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at runtime. If you are not using an Amazon ECS-optimized AMI and you are starting the agent manually with **docker run**, specify the environment variable file with the `--env-file path_to_env_file` option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example-01.com"},"repo.example-02.com":
{"auth":"fQ172MzEXAMPLEoF7225DU0j","email":"email@example-02.com"}}
```

docker authentication format

The `docker` format uses a JSON representation of the registry server that the agent should authenticate with. It also includes the authentication parameters required by that registry (such as user name, password, and the email address for that account). For a Docker Hub account, the JSON representation looks like the following:

```
{
  "https://index.docker.io/v1/": {
    "username": "my_name",
    "password": "my_password",
    "email": "email@example.com"
  }
}
```

In this example, the following environment variables should be added to the environment variable file (`/etc/ecs/ecs.config` for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at runtime. If you are not using an Amazon ECS-optimized AMI, and you are starting the agent manually with **docker run**, specify the environment variable file with the `--env-file` *`path_to_env_file`* option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
```

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"username":"my_name","password":"my_password","email":"email@example-01.com"},"repo.example-02
{"username":"another_name","password":"another_password","email":"email@example-02.com"}}
```

Procedure

Use the following procedure to turn on private registries for your container instances.

To enable private registries in the Amazon ECS-optimized AMI

1. Log in to your container instance using SSH.
2. Open the `/etc/ecs/ecs.config` file and add the `ECS_ENGINE_AUTH_TYPE` and `ECS_ENGINE_AUTH_DATA` values for your registry and account:

   ```
   sudo vi /etc/ecs/ecs.config
   ```

   This example authenticates a Docker Hub user account:

   ```
   ECS_ENGINE_AUTH_TYPE=docker
   ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
   {"username":"my_name","password":"my_password","email":"email@example.com"}}
   ```

3. Check to see if your agent uses the `ECS_DATADIR` environment variable to save its state:

   ```
   docker inspect ecs-agent | grep ECS_DATADIR
   ```

   Output:

   ```
   "ECS_DATADIR=/data",
   ```

> **Important**
>
> If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before stopping the agent. Newer agents with the `ECS_DATADIR` environment variable save their state and you can stop and start them while tasks are running without issues. For more information, see Updating the Amazon ECS container agent.

4. Stop the `ecs` service:

```
sudo stop ecs
```

   Output:

```
ecs stop/waiting
```

5. Restart the `ecs` service.

   - For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

   - For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs && sudo start ecs
```

6. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see the section called "Container introspection".

```
curl http://localhost:51678/v1/metadata
```

Automatic Amazon ECS task and image cleanup

Each time a task is placed on a container instance, the Amazon ECS container agent checks to see if the images referenced in the task are the most recent of the specified tag in the repository. If not, the default behavior allows the agent to pull the images from their respective repositories.

If you frequently update the images in your tasks and services, your container instance storage can quickly fill up with Docker images that you are no longer using and may never use again. For example, you may use a continuous integration and continuous deployment (CI/CD) pipeline.

> **Note**
>
> The Amazon ECS agent image pull behavior can be customized using the `ECS_IMAGE_PULL_BEHAVIOR` parameter. For more information, see Amazon ECS container agent configuration.

Likewise, containers that belong to stopped tasks can also consume container instance storage with log information, data volumes, and other artifacts. These artifacts are useful for debugging containers that have stopped unexpectedly, but most of this storage can be safely freed up after a period of time.

By default, the Amazon ECS container agent automatically cleans up stopped tasks and Docker images that are not being used by any tasks on your container instances.

> **Note**
>
> The automated image cleanup feature requires at least version 1.13.0 of the Amazon ECS container agent. To update your agent to the latest version, see Updating the Amazon ECS container agent.

The following agent configuration variables are available to tune your automated task and image cleanup experience. For more information about how to set these variables on your container instances, see Amazon ECS container agent configuration.

`ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`

The default time to wait to delete containers for a stopped task. If the value is set to less than 1 second, the value is ignored. By default, this parameter is set to 3 hours, but you can reduce this period to as low as 1 second if you need to for your application.

> **Note**
>
> The image cleanup process cannot delete an image as long as there is a container that references it. After containers are removed, any unreferenced images become candidates for cleanup based on the image cleanup configuration parameters.

`ECS_DISABLE_IMAGE_CLEANUP`

If you set this variable to `true`, then automated image cleanup is turned off on your container instance and no images are automatically removed.

`ECS_IMAGE_CLEANUP_INTERVAL`

This variable specifies how frequently the automated image cleanup process should check for images to delete. The default is every 30 minutes but you can reduce this period to as low as 10 minutes to remove images more frequently.

`ECS_IMAGE_MINIMUM_CLEANUP_AGE`

This variable specifies the minimum amount of time between when an image was pulled and when it may become a candidate for removal. This is used to prevent cleaning up images that have just been pulled. The default is 1 hour.

`ECS_NUM_IMAGES_DELETE_PER_CYCLE`

This variable specifies how many images may be removed during a single cleanup cycle. The default is 5 and the minimum is 1.

When the Amazon ECS container agent is running and automated image cleanup is not turned off, the agent checks for Docker images that are not referenced by running or stopped containers at a frequency determined by the `ECS_IMAGE_CLEANUP_INTERVAL` variable. If unused images are found and they are older than the minimum cleanup time specified by the `ECS_IMAGE_MINIMUM_CLEANUP_AGE` variable, the agent removes up to the maximum number of images that are specified with the `ECS_NUM_IMAGES_DELETE_PER_CYCLE` variable. The least-recently referenced images are deleted first. After the images are removed, the agent waits until the next interval and repeats the process again.

Schedule your containers on Amazon ECS

Amazon Elastic Container Service (Amazon ECS) is a shared state, optimistic concurrency system that provides flexible scheduling capabilities for your containerized workloads. The Amazon ECS schedulers use the same cluster state information as the Amazon ECS API to make appropriate placement decisions.

Amazon ECS provides a service scheduler for long-running tasks and applications. It also provides the ability to run standalone tasks or scheduled tasks for batch jobs or single run tasks. You can specify the task placement strategies and constraints for running tasks that best meet your needs. For example, you can specify whether tasks run across multiple Availability Zones or within a single Availability Zone. And, optionally, you can integrate tasks with your own custom or third-party schedulers.

Option	When to use	More information
Service	The *service scheduler* is suitable for long running stateless services and applications. The service scheduler optionally also makes sure that tasks are registered against an Elastic Load Balancing load balancer. You can update your services that are maintained by the service scheduler. This might include deploying a new task definition or changing the number of desired tasks that are running. By default, the service scheduler spreads tasks across multiple Availabil ity Zones. However, you can use task placement strategies	Amazon ECS services

Option	When to use	More information
	and constraints to customize task placement decisions.	
Standalone task	A standalone task is suitable for processes such as batch jobs that perform work and then stop. For example, you can have a process call `RunTask` when work comes into a queue. The task pulls work from the queue, performs the work, and then exits. Using `RunTask`, you can allow the default task placement strategy to distribute tasks randomly across your cluster. This minimizes the chances that a single instance gets a disproportionate number of tasks.	Amazon ECS standalone tasks

Option	When to use	More information
Scheduled tasks	A scheduled task is suitable when you have tasks to run at set intervals in your cluster, you can use EventBridge Scheduler to create a schedule. You can run tasks for a backup operation or a log scan. The EventBridge Scheduler schedule that you create can run one or more tasks in your cluster at specified times. Your scheduled event can be set to a specific interval (run every *N* minutes, hours, or days). Otherwise, for more complicated scheduling, you can use a `cron` expression.	[Using Amazon EventBridge Scheduler to schedule Amazon ECS tasks]()

Compute options

With Amazon ECS, you can specify the infrastructure your tasks or services run on. You can use a capacity provider strategy, or a launch type.

For Fargate, the capacity providers are Fargate and Fargate Spot. For EC2, the capacity provider is the Auto Scaling group with the registered container instances.

The capacity provider strategy distributes your tasks across the capacity providers associated with your cluster.

Only capacity providers that are both already associated with a cluster and have an `ACTIVE` or `UPDATING` status can be used in a capacity provider strategy. You can associate a capacity provider with a cluster when you create a cluster.

In a capacity provider strategy, the optional *base* value designates how many tasks, at a minimum, run on a specified capacity provider. Only one capacity provider in a capacity provider strategy can have a base defined.

The *weight* value determines the relative percentage of the total number of launched tasks that use the specified capacity provider. Consider the following example. You have a strategy that contains two capacity providers, and both have a weight of 1. When the base percentage is reached, the tasks are split evenly across the two capacity providers. Using that same logic, suppose that you specify a weight of 1 for *capacityProviderA* and a weight of 4 for *capacityProviderB*. Then, for every one task that's run using *capacityProviderA*, there are four tasks that use *capacityProviderB*.

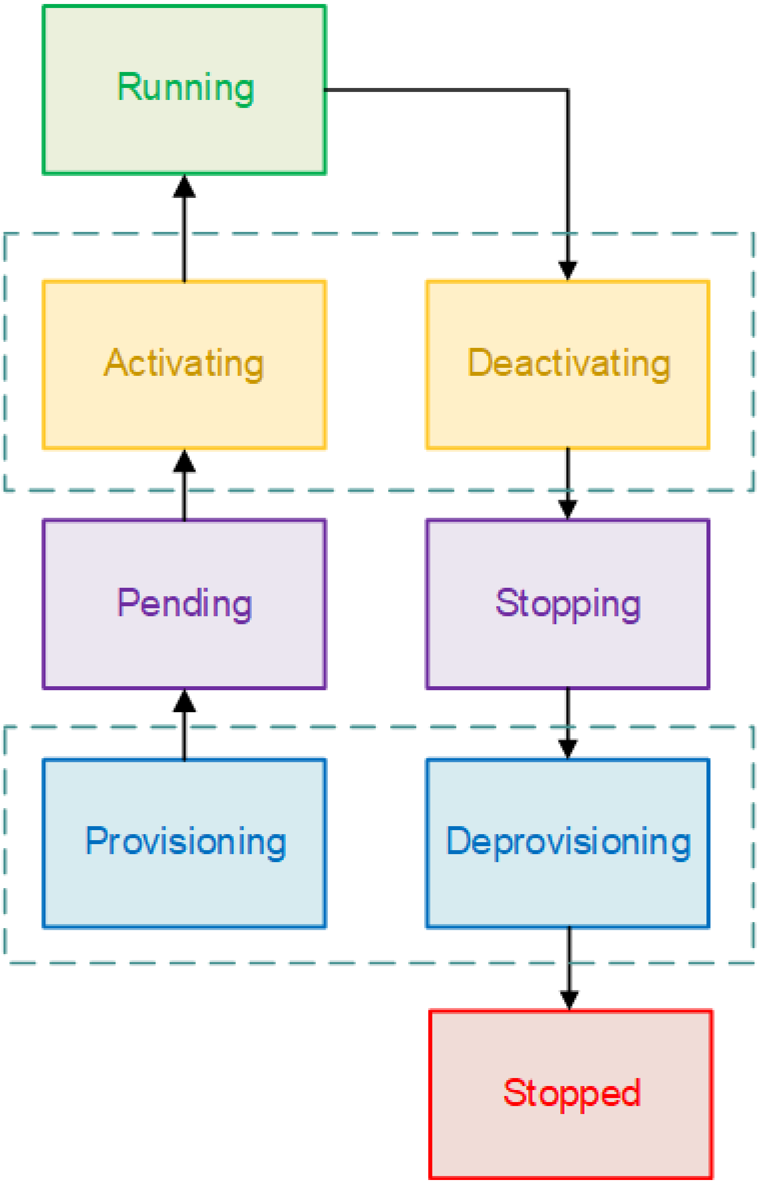
The launch type launches your tasks directly on either Fargate or on the Amazon EC2 instances that you have manually registered to your clusters.

Amazon ECS task lifecycle

When a task is started, either manually or as part of a service, it can pass through several states before it finishes on its own or is stopped manually. Some tasks are meant to run as batch jobs that naturally progress through from `PENDING` to `RUNNING` to `STOPPED`. Other tasks, which can be part of a service, are meant to continue running indefinitely, or to be scaled up and down as needed.

When task status changes are requested, such as stopping a task or updating the desired count of a service to scale it up or down, the Amazon ECS container agent tracks these changes as the last known status (`lastStatus`) of the task and the desired status (`desiredStatus`) of the task. Both the last known status and desired status of a task can be seen either in the console or by describing the task with the API or AWS CLI.

The flow chart below shows the task lifecycle flow.



Lifecycle states

The following are descriptions of each of the task lifecycle states.

PROVISIONING

Amazon ECS has to perform additional steps before the task is launched. For example, for tasks that use the `awsvpc` network mode, the elastic network interface needs to be provisioned.

PENDING

This is a transition state where Amazon ECS is waiting on the container agent to take further action. A task stays in the pending state until there are available resources for the task.

ACTIVATING

This is a transition state where Amazon ECS has to perform additional steps after the task is launched but before the task can transition to the `RUNNING` state. This is the state where Amazon ECS pulls the container images, creates the containers, configures the task networking, registers load balancer target groups, and configures service discovery.

RUNNING

The task is successfully running.

DEACTIVATING

This is a transition state where Amazon ECS has to perform additional steps before the task is stopped. For example, for tasks that are part of a service that's configured to use Elastic Load Balancings target groups, the target group deregistration occurs during this state.

STOPPING

This is a transition state where Amazon ECS is waiting on the container agent to take further action.

For Linux containers, the container agent will send the `SIGTERM` signal to notify the application needs to finish and shut down, and then send a `SIGKILL` after waiting the `StopTimeout` duration set in the task definition.

DEPROVISIONING

Amazon ECS has to perform additional steps after the task has stopped but before the task transitions to the `STOPPED` state. For example, for tasks that use the `awsvpc` network mode, the elastic network interface needs to be detached and deleted.

STOPPED

The task has been successfully stopped.

If your task stopped because of an error, see Viewing Amazon ECS stopped task errors.

DELETED

This is a transition state when a task stops. This state is not displayed in the console, but is displayed in `describe-tasks`.

How Amazon ECS places tasks on container instances

You can use task placement to configure Amazon ECS to place your tasks on container instances that meet certain criteria, for example an Availability Zone or instance type.

The following are task placement components:

- Task placement strategy - The algorithm for selecting container instances for task placement or tasks for termination. For example, Amazon ECS can select container instances at random, or it can select container instances such that tasks are distributed evenly across a group of instances.
- Task group - A group of related tasks, for example database tasks.
- Task placement constraint - These are rules that must be met in order to place a task on a container instance. If the constraint is not met, the task is not placed and remains in the `PENDING` state. For example, you can use a constraint to place tasks only on a particular instance type.

Amazon ECS has different algorithms for the launch types.

EC2 launch type

For tasks that use the EC2 launch type, Amazon ECS must determine where to place the task based on the requirements specified in the task definition, such as CPU and memory. Similarly, when you scale down the task count, Amazon ECS must determine which tasks to terminate. You can apply task placement strategies and constraints to customize how Amazon ECS places and terminates tasks.

The default task placement strategies depend on whether you run tasks manually (standalone tasks) or within a service. For tasks running as part of an Amazon ECS service, the task placement strategy is `spread` using the `attribute:ecs.availability-zone`. There isn't a default task placement constraint for tasks not in services. For more information, see Schedule your containers on Amazon ECS.

> **Note**
>
> Task placement strategies are a best effort. Amazon ECS still attempts to place tasks even when the most optimal placement option is unavailable. However, task placement constraints are binding, and they can prevent task placement.

You can use task placement strategies and constraints together. For example, you can use a task placement strategy and a task placement constraint to distribute tasks across Availability Zones and bin pack tasks based on memory within each Availability Zone, but only for G2 instances.

When Amazon ECS places tasks, it uses the following process to select container instances:

1. Identify the container instances that satisfy the CPU, GPU, memory, and port requirements in the task definition.
2. Identify the container instances that satisfy the task placement constraints.
3. Identify the container instances that satisfy the task placement strategies.
4. Select the container instances for task placement.

Fargate launch type

Task placement strategies and constraints aren't supported for tasks using the Fargate launch type. Fargate will try its best to spread tasks across accessible Availability Zones. If the capacity provider includes both Fargate and Fargate Spot, the spread behavior is independent for each capacity provider.

Use strategies to define Amazon ECS task placement

For tasks that use the EC2 launch type, Amazon ECS must determine where to place the task based on the requirements specified in the task definition, such as CPU and memory. Similarly, when you scale down the task count, Amazon ECS must determine which tasks to terminate. You can apply task placement strategies and constraints to customize how Amazon ECS places and terminates tasks.

The default task placement strategies depend on whether you run tasks manually (standalone tasks) or within a service. For tasks running as part of an Amazon ECS service, the task placement strategy is `spread` using the `attribute:ecs.availability-zone`. There isn't a default task placement constraint for tasks not in services. For more information, see [Schedule your containers on Amazon ECS](#).

> **Note**
>
> Task placement strategies are a best effort. Amazon ECS still attempts to place tasks even when the most optimal placement option is unavailable. However, task placement constraints are binding, and they can prevent task placement.

You can use task placement strategies and constraints together. For example, you can use a task placement strategy and a task placement constraint to distribute tasks across Availability Zones and bin pack tasks based on memory within each Availability Zone, but only for G2 instances.

When Amazon ECS places tasks, it uses the following process to select container instances:

1. Identify the container instances that satisfy the CPU, GPU, memory, and port requirements in the task definition.
2. Identify the container instances that satisfy the task placement constraints.
3. Identify the container instances that satisfy the task placement strategies.
4. Select the container instances for task placement.

You specify task placement strategies in the service definition, or task definition using the `placementStrategy` parameter.

```
"placementStrategy": [
    {
        "field": "The field to apply the placement strategy against",
        "type": "The placement strategy to use"
    }
]
```

You can specify the strategies when you run a task (RunTask), create a new service (CreateService), or update an existing service (UpdateService).

The following table describes the available types and fields.

type	Valid field values	
`binpack`	• cpu • memory	

type	Valid field values	
Tasks are placed on container instances so as to leave the least amount of unused CPU or memory. This strategy minimizes the number of container instances in use. When this strategy is used and a scale-in action is taken, Amazon ECS terminates tasks. It does this based on the amount of resources that are left on the container instance after the task is terminated. The container instance that has the most available resources left after task termination has that task terminated.		
`random` Tasks are placed randomly.	Not used	

type	Valid field values	
`spread` Tasks are placed evenly based on the specified value. Service tasks are spread based on the tasks from that service. Standalone tasks are spread based on the tasks from the same task group. For more information about task groups, see Group related Amazon ECS tasks . When the `spread` strategy is used and a scale-in action is taken, Amazon ECS selects tasks to terminate that maintain a balance across Availability Zones. Within an Availability Zone, tasks are selected at random.	• `instanceId` (or `host`, which has the same effect) • any platform or custom attribute that's applied to a container instance, such as `attribute:ecs.availability-zone`	

The task placement strategies can be updated for existing services as well. For more information, see How Amazon ECS places tasks on container instances.

You can create a task placement strategy that uses multiple strategies by creating arrays of strategies in the order that you want them performed. For example, if you want to spread tasks across Availability Zones and then bin pack tasks based on memory within each Availability Zone, specify the Availability Zone strategy followed by the memory strategy. For example strategies, see Example Amazon ECS task placement strategies.

Example Amazon ECS task placement strategies

You can specify task placement strategies with the following actions: CreateService, UpdateService, and RunTask.

Examples

- Distribute tasks evenly across Availability Zones
- Distribute tasks evenly across all instances
- Bin pack tasks based on memory
- Place tasks randomly
- Distribute tasks evenly across Availability Zones and then distributes tasks evenly across the instances within each Availability Zone
- Distribute tasks evenly across Availability Zones and then bin pack tasks based on memory within each Availability Zone
- Distribute tasks evenly across instances and then bin pack tasks based on memory

Distribute tasks evenly across Availability Zones

The following strategy distributes tasks evenly across Availability Zones.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    }
]
```

Distribute tasks evenly across all instances

The following strategy distributes tasks evenly across all instances.

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

Bin pack tasks based on memory

The following strategy bin packs tasks based on memory.

```
"placementStrategy": [
    {
```

```
        "field": "memory",
        "type": "binpack"
    }
]
```

Place tasks randomly

The following strategy places tasks randomly.

```
"placementStrategy": [
    {
        "type": "random"
    }
]
```

Distribute tasks evenly across Availability Zones and then distributes tasks evenly across the instances within each Availability Zone

The following strategy distributes tasks evenly across Availability Zones and then distributes tasks evenly across the instances within each Availability Zone.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

Distribute tasks evenly across Availability Zones and then bin pack tasks based on memory within each Availability Zone

The following strategy distributes tasks evenly across Availability Zones and then bin packs tasks based on memory within each Availability Zone.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
```

```
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

Distribute tasks evenly across instances and then bin pack tasks based on memory

The following strategy distributes tasks evenly across evenly across all instances and then bin packs tasks based on memory within each instance.

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

Group related Amazon ECS tasks

You can identify a set of related tasks and place them in a task group. All tasks with the same task group name are considered as a set when using the `spread` task placement strategy. For example, suppose that you're running different applications in one cluster, such as databases and web servers. To ensure that your databases are balanced across Availability Zones, add them to a task group named `databases` and then use the `spread` task placement strategy. For more information, see Use strategies to define Amazon ECS task placement.

Task groups can also be used as a task placement constraint. When you specify a task group in the `memberOf` constraint, tasks are only sent to container instances that run tasks in the specified task group. For an example, see Example Amazon ECS task placement constraints.

By default, standalone tasks use the task definition family name (for example, `family:my-task-definition`) as the task group name if a custom task group name isn't specified. Tasks launched as part of a service use the service name as the task group name and can't be changed.

The following requirements for the task group apply.

- A task group name must be 255 or fewer characters.
- Each task can be in exactly one group.
- After launching a task, you can't modify its task group.

Define which container instances Amazon ECS uses for tasks

A task placement constraint is a rule about a container instance that Amazon ECS uses to determine if the task is allowed to run on the instance. At least one container instance must match the constraint. If there are no instances that match the constraint, the task remains in a `PENDING` state. When you create a new service or update an existing one, you can specify task placement constraints for the service's tasks.

You can specify task placement constraints in the service definition, task definition, or task using the `placementConstraint` parameter.

```
"placementConstraints": [
    {
        "expression": "The expression that defines the task placement constraints",
        "type": "The placement constraint type to use"
    }
]
```

The following table describes how to use the parameters.

Constraint type	Can be specified when	
`distinctInstance` Place each active task on a different container instance. Amazon ECS looks at the desired status of the tasks for the task placement. For example, if the desired status of the existing task is `STOPPED`, (but the last status isn't), a new incoming	• Running a task RunTask • Creating a new service CreateService,	

Constraint type	Can be specified when	
task can be placed on the same instance despite the `distinctInstance` placement constraint. Therefore, you might see 2 tasks with last status of `RUNNING` on the same instance. **Important** We recommend that customers looking for strong isolation for their tasks use Fargate. Fargate runs each task in a hardware virtualiz ation environment. This ensures that these container ized workloads do not share network interfaces, Fargate ephemeral storage, CPU, or memory with other tasks. For more information, see [Security Overview of AWS Fargate]().		

Constraint type	Can be specified when	
`memberOf` Place tasks on container instances that satisfy an expression.	• Running a task RunTask • Creating a new service CreateService, • Creating a new task definition RegisterTaskDefinition • Creating a new revision of a task definition RegisterTaskDefinition • Updating a service UpdateService	

When you use the `memberOf` constraint type, you can create an expression using the cluster query language which defines the container instances where Amazon ECS can place tasks. The expression is a way for you to group your container instances by attributes. The expression goes in the `expression` parameter of `placementConstraint`.

Amazon ECS container instance attributes

You can add custom metadata to your container instances, known as *attributes*. Each attribute has a name and an optional string value. You can use the built-in attributes provided by Amazon ECS or define custom attributes.

The following sections contain sample built-in, optional, and custom attributes.

Built-in attributes

Amazon ECS automatically applies the following attributes to your container instances.

`ecs.ami-id`

The ID of the AMI used to launch the instance. An example value for this attribute is `ami-1234abcd`.

`ecs.availability-zone`

The Availability Zone for the instance. An example value for this attribute is `us-east-1a`.

`ecs.instance-type`

The instance type for the instance. An example value for this attribute is `g2.2xlarge`.

`ecs.os-type`

The operating system for the instance. The possible values for this attribute are `linux` and `windows`.

`ecs.os-family`

The operating system version for the instance.

For Linux instances, the valid value is `LINUX`. For Windows instances, ECS sets the value in the `WINDOWS_SERVER_<OS_Release>_<FULL or CORE>` format. The valid values are `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_20H2_CORE`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE`, and `WINDOWS_SERVER_2016_FULL`.

This is important for Windows containers and Windows containers on AWS Fargate because the OS version of every Windows container must match that of the host. If the Windows version of the container image is different than the host, the container doesn't start. For more information, see Windows container version compatibility on the Microsoft documentation website.

If your cluster runs multiple Windows versions, you can ensure that a task is placed on an EC2 instance running on the same version by using the placement constraint: `memberOf(attribute:ecs.os-family == WINDOWS_SERVER_<OS_Release>_<FULL or CORE>)`. For more information, see the section called “Retrieving Amazon ECS-optimized Windows AMI metadata”.

`ecs.cpu-architecture`

The CPU architecture for the instance. Example values for this attribute are `x86_64` and `arm64`.

`ecs.vpc-id`

The VPC the instance was launched into. An example value for this attribute is `vpc-1234abcd`.

`ecs.subnet-id`

The subnet the instance is using. An example value for this attribute is `subnet-1234abcd`.

Optional attributes

Amazon ECS may add the following attributes to your container instances.

`ecs.awsvpc-trunk-id`

If this attribute exists, the instance has a trunk network interface. For more information, see Increasing Amazon ECS Linux container instance network interfaces.

`ecs.outpost-arn`

If this attribute exists, it contains the Amazon Resource Name (ARN) of the Outpost. For more information, see the section called "Amazon Elastic Container Service on AWS Outposts".

`ecs.capability.external`

If this attribute exists, the instance is identified as an external instance. For more information, see Amazon ECS clusters for the external launch type.

Custom attributes

You can apply custom attributes to your container instances. For example, you can define an attribute with the name "stack" and a value of "prod".

When specifying custom attributes, you must consider the following.

- The `name` must contain between 1 and 128 characters and name may contain letters (uppercase and lowercase), numbers, hyphens, underscores, forward slashes, back slashes, or periods.
- The `value` must contain between 1 and 128 characters and may contain letters (uppercase and lowercase), numbers, hyphens, underscores, periods, at signs (@), forward slashes, back slashes, colons, or spaces. The value can't contain any leading or trailing whitespace.

Create expressions to define container instances for Amazon ECS tasks

Cluster queries are expressions that allow you to group objects. For example, you can group container instances by attributes such as Availability Zone, instance type, or custom metadata. For more information, see Amazon ECS container instance attributes.

After you have defined a group of container instances, you can customize Amazon ECS to place tasks on container instances based on group. For more information, see Running an application as an Amazon ECS task, and Creating an Amazon ECS rolling update deployment. You can also apply a group filter when listing container instances.

Expression syntax

Expressions have the following syntax:

```
subject operator [argument]
```

Subject

The attribute or field to be evaluated.

`agentConnected`

Select container instances by their Amazon ECS container agent connection status. You can use this filter to search for instances with container agents that are disconnected.

Valid operators: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`agentVersion`

Select container instances by their Amazon ECS container agent version. You can use this filter to find instances that are running outdated versions of the Amazon ECS container agent.

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`attribute:attribute-name`

Select container instances by attribute. For more information, see Amazon ECS container instance attributes.

`ec2InstanceId`

Select container instances by their Amazon EC2 instance ID.

Valid operators: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`registeredAt`

Select container instances by their container instance registration date. You can use this filter to find newly registered instances or instances that are very old.

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

Valid date formats: 2018-06-18T22:28:28+00:00, 2018-06-18T22:28:28Z, 2018-06-18T22:28:28, 2018-06-18

`runningTasksCount`

Select container instances by number of running tasks. You can use this filter to find instances that are empty or near empty (few tasks running on them).

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`task:group`

Select container instances by task group. For more information, see Group related Amazon ECS tasks .

Operator

The comparison operator. The following operators are supported.

Operator	Description
==, equals	String equality
!=, not_equals	String inequality
>, greater_than	Greater than
>=, greater_than_equal	Greater than or equal to
<, less_than	Less than
<=, less_than_equal	Less than or equal to
exists	Subject exists
!exists, not_exists	Subject doesn't exist
in	Value in argument list

Operator	Description
!in, not_in	Value not in argument list
=~, matches	Pattern match
!~, not_matches	Pattern mismatch

Note

A single expression can't contain parentheses. However, parentheses can be used to specify precedence in compound expressions.

Argument

For many operators, the argument is a literal value.

The `in` and `not_in` operators expect an argument list as the argument. You specify an argument list as follows:

```
[argument1, argument2, ..., argumentN]
```

The matches and not_matches operators expect an argument that conforms to the Java regular expression syntax. For more information, see java.util.regex.Pattern.

Compound expressions

You can combine expressions using the following Boolean operators:

- &&, and
- ||, or
- !, not

You can specify precedence using parentheses:

```
(expression1 or expression2) and expression3
```

Example expressions

The following are example expressions.

Example: String Equality

The following expression selects instances with the specified instance type.

```
attribute:ecs.instance-type == t2.small
```

Example: Argument List

The following expression selects instances in the us-east-1a or us-east-1b Availability Zone.

```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

Example: Compound Expression

The following expression selects G2 instances that aren't in the us-east-1d Availability Zone.

```
attribute:ecs.instance-type =~ g2.* and attribute:ecs.availability-zone != us-east-1d
```

Example: Task Affinity

The following expression selects instances that are hosting tasks in the `service:production` group.

```
task:group == service:production
```

Example: Task Anti-Affinity

The following expression selects instances that aren't hosting tasks in the database group.

```
not(task:group == database)
```

Example: Running task count

The following expression selects instances that are only running one task.

```
runningTasksCount == 1
```

Example: Amazon ECS container agent version

The following expression selects instances that are running a container agent version below 1.14.5.

```
agentVersion < 1.14.5
```

Example: Instance registration time

The following expression selects instances that were registered before February 13, 2018.

```
registeredAt < 2018-02-13
```

Example: Amazon EC2 instance ID

The following expression selects instances with the following Amazon EC2 instance IDs.

```
ec2InstanceId in ['i-abcd1234', 'i-wxyx7890']
```

Example Amazon ECS task placement constraints

The following are task placement constraint examples.

This example uses the `memberOf` constraint to place tasks on t2 instances. It can be specified with the following actions: CreateService, UpdateService, RegisterTaskDefinition, and RunTask.

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-type =~ t2.*",
        "type": "memberOf"
    }
]
```

The example uses the `memberOf` constraint to place replica tasks on instances with tasks in the daemon service `daemon-service` task group, respecting any task placement strategies that are also specified. This constraint ensures that the daemon service tasks get placed on the EC2 instance prior to the replica service tasks.

Replace `daemon-service` with the name of the daemon service.

```
"placementConstraints": [
    {
```

```
        "expression": "task:group == service:daemon-service",
        "type": "memberOf"
    }
]
```

The example uses the memberOf constraint to place tasks on instances with other tasks in the databases task group, respecting any task placement strategies that are also specified. For more information about task groups, see Group related Amazon ECS tasks . It can be specified with the following actions: CreateService, UpdateService, RegisterTaskDefinition, and RunTask.

```
"placementConstraints": [
    {
        "expression": "task:group == databases",
        "type": "memberOf"
    }
]
```

The distinctInstance constraint places each task in the group on a different instance. It can be specified with the following actions: CreateService, UpdateService, and RunTask

Amazon ECS looks at the desired status of the tasks for the task placement. For example, if the desired status of the existing task is STOPPED, (but the last status isn't), a new incoming task can be placed on the same instance despite the distinctInstance placement constraint. Therefore, you might see 2 tasks with last status of RUNNING on the same instance.

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

Amazon ECS standalone tasks

You can run your application as a task when you have an application that performs some work, and then stops, for example a batch process. If you want to run a task one time, you can use the console, AWS CLI, APIs, or SDKs.

If you need to run your application on a rate-based, cron-based, or one-time schedule, you can create schedule using EventBridge Scheduler.

Task workflow

When you launch Amazon ECS tasks (standalone tasks or by Amazon ECS services), a task is created and initially moved to the `PROVISIONING` state. When a task is in the `PROVISIONING` state, neither the task nor the containers exist because Amazon ECS needs to find compute capacity for placing the task.

Amazon ECS selects the appropriate compute capacity for your task based on your launch type or capacity provider configuration. You can use capacity providers and capacity provider strategies with both the Fargate and Amazon EC2 launch types. With Fargate, you don't have to think about provisioning, configuring, and scaling of your cluster capacity. Fargate takes care of all infrastructure management for your tasks. For the EC2 launch type, you can either manage your cluster capacity by registering Amazon EC2 instances to your cluster, or you can use cluster auto scaling to simplify your compute capacity management. Cluster auto scaling takes care of dynamically scaling your cluster capacity, so that you can focus on running tasks. Amazon ECS determines where to place the task based on the requirements you specify in the task definition, such as CPU and memory, as well your placement constraints and strategies. For more information, see, How Amazon ECS places tasks on container instances.

If you use a capacity provider with managed scaling enabled, tasks that can't be started due to a lack of compute capacity are moved to the `PROVISIONING` state rather than failing immediately. After finding the capacity for placing your task, Amazon ECS provisions the necessary attachments (for example, Elastic Network Interfaces (ENIs) for tasks in `awsvpc` mode). It uses the Amazon ECS container agent to pull your container images, and then start your containers. After the provisioning completes and the relevant containers have launched, Amazon ECS moves the task into `RUNNING` state. For information about the task states, see Amazon ECS task lifecycle.

Running an application as an Amazon ECS task

You can create a task for a one-time process using the AWS Management Console.

To create a standalone task (AWS Management Console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. The Amazon ECS console allows you to create a standalone task from either your cluster detail page or from the task definition revision list. Use the following steps to create your standalone task depending on the resource page you choose.

To start a service from	Steps	
The cluster detail page	a. On the **Clusters** page, select the cluster to create the service in. b. From the **Tasks** tab, choose **Run task**.	
The task definition revision page	a. On the **Task definitio ns** page, choose the task definition family to display the revisions for that family. b. Select the revision you want to use. c. From the **Deploy** menu, choose **Run task**.	

3. For **Existing cluster**, choose the cluster.

   Choose **Create cluster** to run the task on a new cluster

4. Choose how your tasks are distributed across your cluster infrastructure. Under **Compute configuration**, choose your option.To use a capacity provider strategy, you must configure your capacity providers at the cluster level.

   If you haven't configured your cluster to use a capacity provider, use a launch type instead.

Distribution method	Steps	
Capacity provider strategy	a. In the **Compute options** section, select **Capacity provider strategy**. b. Choose a strategy: • To use the cluster's default capacity	

Distribution method	Steps	
	provider strategy, choose **Use cluster default**. • If your cluster doesn't have a default capacity provider strategy, or to use a custom strategy, choose **Use custom**, **Add capacity provider strategy** and define your custom capacity provider strategy by specifying a **Base**, **Capacity provider**, and **Weight**. **Note** To use a capacity provider in a strategy, the capacity provider must be associated with the cluster.	

Distribution method	Steps	
Launch type	a. In the **Compute options** section, select **Launch type**. b. For **Launch type**, choose a launch type. c. (Optional) When the Fargate launch type is specified, for **Platform version**, specify the platform version to use. If a platform version isn't specified, the `LATEST` platform version is used.	

5. Under **Deployment configuration**, do the following:

   a. For **Task definition**, enter the task definition.

   > **Important**
   >
   > The console validates the selection to ensure that the selected task definition family and revision are compatible with the defined compute configuration.

   b. For **Desired tasks**, enter the number of tasks to launch.

   c. For **Task group**, enter the task group name.

6. If your task definition uses the `awsvpc` network mode, expand **Networking**. Use the following steps to specify a custom configuration.

   a. For **VPC**, select the VPC to use.

   b. For **Subnets**, select one or more subnets in the VPC that the task scheduler considers when placing your tasks.

   c. For **Security group**, you can either choose an existing security group or create a new one. To use an existing security group, choose the security group and move to the next step. To create a new security group, choose **Create a new security group**. You must specify a security group name, description, and then add one or more inbound rules for the security group.

d. For **Public IP**, choose whether to auto-assign a public IP address to the elastic network interface (ENI) of the task.

   AWS Fargate tasks can be assigned a public IP address when run in a public subnet so they have a route to the internet. EC2 tasks can't be assigned a public IP using this field. For more information, see Amazon ECS task networking options for the Fargate launch type and Allocate a network interface for an Amazon ECS task..

7. If your task uses a data volume that's compatible with configuration at deployment, you can configure the volume by expanding **Volume**.

   The volume name and volume type are configured when creating a task definition revision and can't be changed when you run a standalone task. To update the volume name and type, you must create a new task definition revision and run a task by using the new revision.

To configure this volume type	Do this	
Amazon EBS	a. For **EBS volume type**, choose the type of EBS volume that you want to attach to your task. b. For **Size (GiB)**, enter a valid value for the volume size in gibibytes (GiB). You can specify a minimum of 1 GiB and a maximum of 16,384 GiB volume size. This value is required unless you provide a snapshot ID. c. For **IOPS**, enter the maximum number of input/output operation s (IOPS) that the volume	

To configure this volume type	Do this	
	should provide. This value is configurable only for `io1`,`io2`, and `gp3` volume types. d. For **Throughput (MiB/s)**, enter the throughput that the volume should provide, in mebibytes per second (MiBps, or MiB/s). This value is configurable only for the `gp3` volume type. e. For **Snapshot ID**, choose an existing Amazon EBS volume snapshot or enter the ARN of a snapshot if you want to create a volume from a snapshot. You can also create a new, empty volume by not choosing or entering a snapshot ID. f. If you specify a **Snapshot ID**, you can specify a **Volume initialization rate (MiB/s)**. Enter a value between 100 and 300, in MiB/s, that will determine how fast data is loaded from the snapshot specified using **Snapshot ID** for volume creation.	

To configure this volume type	Do this	
	g. For **Termination policy**, deselect the checkbox if you want the volume configured for attachment to the task to be preserved after the task is terminated. By default, EBS volumes that are attached to tasks are deleted when the task is terminated. h. For **File system type**, choose the type of file system that will be used for data storage and retrieval on the volume. You can choose either the operating system default or a specific file system type. The default for Linux is XFS. For volumes created from a snapshot, you must specify the same filesystem type that the volume was using when the snapshot was created. If there is a filesystem type mismatch, the task will fail to start. i. For **Infrastructure role**, choose an IAM role with the necessary permissio	

To configure this volume type	Do this	
	ns that allow Amazon ECS to manage Amazon EBS volumes for tasks. You can attach the `AmazonECSInfrastructureRolePolicyForVolumes` managed policy to the role, or you can use the policy as a guide to create and attach an your own policy with permissions that meet your specific needs. For more information about the necessary permissions, see see [Amazon ECS infrastructure IAM role](). j. For **Encryption**, choose **Default** if you want to use the Amazon EBS encryption by default settings. If your account has [Encryption by default]() configured, the volume will be encrypted with the AWS Key Management Service (AWS KMS) key that's specified in the setting. If you choose **Default** and Amazon EBS default encryption isn't turned on, the volume will be unencrypted.	

To configure this volume type	Do this	
	If you choose **Custom**, you can specify an AWS KMS key of your choice for volume encryption. If you choose **None**, the volume will be unencrypted unless you have encryption by default configured, or if you create a volume from an encrypted snapshot. k. If you've chosen **Custom** for **Encryption**, you must specify the AWS KMS key that you want to use. For **KMS key**, choose an AWS KMS key or enter a key ARN. If you choose to encrypt your volume by using a symmetric customer managed key, make sure that you have the right permissions defined in your AWS KMS key policy. For more information, see [Data encryption for Amazon EBS volumes](#). l. (Optional) Under **Tags**, you can add tags to your Amazon EBS volume by either propagating tags	

To configure this volume type	Do this	
	from the task definition or by providing your own tags. If you want to propagate tags from the task definition, choose **Task definition** for **Propagate tags from**. If you choose **Do not propagate**, or if you don't choose a value, the tags aren't propagate d. If you want to provide your own tags, choose **Add tag** and then provide the key and value for each tag you add. For more information about tagging Amazon EBS volumes, see [Tagging Amazon EBS volumes].	

8. (Optional) To use a task placement strategy other than the default, expand **Task Placement**, and then choose from the following options.

   For more information, see [How Amazon ECS places tasks on container instances].

   - **AZ Balanced Spread –** Distribute tasks across Availability Zones and across container instances in the Availability Zone.
   - **AZ Balanced BinPack –** Distribute tasks across Availability Zones and across container instances with the least available memory.
   - **BinPack –** Distribute tasks based on the least available amount of CPU or memory.

- **One Task Per Host** – Place, at most, one task from the service on each container instance.
- **Custom** – Define your own task placement strategy.

If you chose **Custom**, define the algorithm for placing tasks and the rules that are considered during task placement.

- Under **Strategy**, for **Type** and **Field**, choose the algorithm and the entity to use for the algorithm.

  You can enter a maximum of 5 strategies.
- Under **Constraint**, for **Type** and **Expression**, choose the rule and attribute for the constraint.

  For example, to set the constraint to place tasks on T2 instances, for the **Expression**, enter **attribute:ecs.instance-type =~ t2.***.

  You can enter a maximum of 10 constraints.

9. (Optional) To override the task IAM role, or task execution role that is defined in your task definition, expand **Task overrides**, and then complete the following steps:

   a. For **Task role**, choose an IAM role for this task. For more information, see Amazon ECS task IAM role.

      Only roles with the `ecs-tasks.amazonaws.com` trust relationship are displayed. For instructions on how to create an IAM role for your tasks, see Creating the task IAM role.

   b. For **Task execution role**, choose a task execution role. For more information, see Amazon ECS task execution IAM role.

10. (Optional) To override the container commands and environment variables, expand **Container Overrides**, and then expand the container.

    - To send a command to the container other than the task definition command, for **Command override**, enter the Docker command.
    - To add an environment variable, choose **Add Environment Variable**. For **Key**, enter the name of your environment variable. For **Value**, enter a string value for your environment value (without the surrounding double quotation marks (" ")).

      AWS surrounds the strings with double quotation marks (" ") and passes the string to the container in the following format:

```
MY_ENV_VAR="This variable contains a string."
```

11. (Optional) To help identify your task, expand the **Tags** section, and then configure your tags.

    To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the task definition tags, select **Turn on Amazon ECS managed tags**, and then select **Task definitions**.

    Add or remove a tag.

    - [Add a tag] Choose **Add tag**, and then do the following:
      - For **Key**, enter the key name.
      - For **Value**, enter the key value.
    - [Remove a tag] Next to the tag, choose **Remove tag**.
12. Choose **Create**.

Using Amazon EventBridge Scheduler to schedule Amazon ECS tasks

EventBridge Scheduler is a serverless scheduler that allows you to create, run, and manage tasks from one central, managed service. It provides one-time and recurring scheduling functionality independent of event buses and rules. EventBridge Scheduler is highly customizable, and offers improved scalability over EventBridge scheduled rules, with a wider set of target API operations and AWS services. EventBridge Scheduler provides the following schedules which you can configure for your tasks in the EventBridge Scheduler console:

- Rate-based
- Cron-based

  You can configure cron-based schedules in any time zone.
- One-time schedules

  You can configure one-time schedules in any time zone.

You can schedule your Amazon ECS using Amazon EventBridge Scheduler.

Although you can create a scheduled task in the Amazon ECS console, currently the EventBridge Scheduler console provides more functionality.

Complete the following steps before you schedule a task:

1. Use the VPC console to get the subnet IDs where the tasks run and the security group IDs for the subnets. For more information, see Subnets for your VPC, and Control traffic to your AWS resources using security groups in the *Amazon VPC User Guide*.
2. Configure the EventBridge Scheduler execution role. For more information, see Set up the execution role in the *Amazon EventBridge Scheduler User Guide*.
3. If you want to use a capacity provider strategy to run the task, you must have a capacity provider associated with the cluster.

To create a new schedule using the console

1. Open the Amazon EventBridge Scheduler console at https://console.aws.amazon.com/scheduler/home.
2. On the **Schedules** page, choose **Create schedule**.
3. On the **Specify schedule detail** page, in the **Schedule name and description** section, do the following:
   a. For **Schedule name**, enter a name for your schedule. For example, **`MyTestSchedule`**.
   b. (Optional) For **Description**, enter a description for your schedule. For example, **`TestSchedule`**.
   c. For **Schedule group**, choose a schedule group. If you don't have a group, choose **default**. To create a schedule group, choose **create your own schedule**.

      You use schedule groups to add tags to groups of schedules.
4. Choose your schedule options.

Occurrence	Do this...	
One-time schedule A one-time schedule invokes a target only once at the date and time that you specify.	For **Date and time**, do the following: • Enter a valid date in `YYYY/MM/DD` format. • Enter a timestamp in 24-hour `hh:mm` format.	

Occurrence	Do this...	
	• For **Timezone**, choose the timezone.	

Occurrence	Do this...	
Recurring schedule A recurring schedule invokes a target at a rate that you specify using a **cron** expression or rate expressio n.	a. For **Schedule type**, do one of the following: • To use a cron expressio n to define the schedule, choose **Cron-based schedule** and enter the cron expression. • To use a rate expression to define the schedule, choose **Rate-based schedule** and enter the rate expression. For more informati on about cron and rate expressions, see [Schedule types on EventBridge Scheduler](#) in the *Amazon EventBridge Scheduler User Guide*. b. For **Flexible time window**, choose **Off** to turn off the option, or choose one of the pre-defined time windows. For example, if you choose **15 minutes** and you set a recurring schedule to invoke its target once every hour, the schedule runs within	

Occurrence	Do this...
	15 minutes after the start of every hour.

5. (Optional) If you chose **Recurring schedule** in the previous step, in the **Timeframe** section, do the following:

   a. For **Timezone**, choose a timezone.

   b. For **Start date and time**, enter a valid date in `YYYY/MM/DD` format, and then specify a timestamp in 24-hour `hh:mm` format.

   c. For **End date and time**, enter a valid date in `YYYY/MM/DD` format, and then specify a timestamp in 24-hour `hh:mm` format.

6. Choose **Next**.

7. On the **Select target** page, do the following:

   a. Choose **All APIs**, and then in the search box enter **ECS**.

   b. Select **Amazon ECS**.

   c. In the search box, enter **RunTask**, and then choose **RunTask**.

   d. For **ECS cluster**, choose the cluster.

   e. For **ECS task**, choose the task definition to use for the task.

   f. Choose how your tasks are distributed across your cluster infrastructure. Expand **Compute options**, and then choose one of the following options

Compute option	Steps
Capacity provider strategy	a. Choose **Capacity provider strategy**. b. Choose a strategy: • To use the default capacity provider strategy, choose **Use cluster default**. • To use a custom strategy, choose **Use**

Compute option	Steps
	custom. Then, enter the **Base**, **Capacity provider**, and **Weight**. For EC2, the Capacity provider is the Auto Scaling group.
Launch type	a. Choose **Launch type**. b. For **Launch type**, choose a launch type. c. When the Fargate launch type is specified , for **Platform version**, specify the platform version to use.

g. For **Subnets**, enter the subnet IDs to run the task in.

h. For **Security groups**, enter the security group IDs for the subnet.

i. (Optional) To use a task placement strategy other than the default, expand **Placement constraint**, and then enter the constraints.

   For more information, see How Amazon ECS places tasks on container instances.

j. (Optional) To help identify your tasks, under **Tags** configure your tags.

   To have Amazon ECS automatically tag all newly launched tasks with the task definition tags, select **Enable Amazon ECS managed tags**.

8. Choose **Next**.

9. On the **Settings** page, do the following:

   a. To turn on the schedule, under **Schedule state**, toggle **Enable schedule**.

   b. To configure a retry policy for your schedule, under **Retry policy and dead-letter queue (DLQ)**, do the following:

      - Toggle **Retry**.

- For **Maximum retention time of event**, enter the maximum **hour(s)** and **min(s)** that EventBridge Scheduler must keep an unprocessed event.
- The maximum time is 24 hours.
- For **Maximum retries**, enter the maximum number of times EventBridge Scheduler retries the schedule if the target returns an error.

  The maximum value is 185 retries.

With retry policies, if a schedule fails to invoke its target, EventBridge Scheduler re-runs the schedule. If configured, you must set the maximum retention time and retries for the schedule.

c. Choose where EventBridge Scheduler stores undelivered events.

Dead-letter queue (DLQ) option	Do this...	
Don't store	Choose **None**.	
Store the event in the same AWS account where you're creating the schedule	a. Choose **Select an Amazon SQS queue in my AWS account as a DLQ**. b. Choose the Amazon Resource Name (ARN) of the Amazon SQS queue.	
Store the event in a different AWS account from where you're creating the schedule	a. Choose **Specify an Amazon SQS queue in other AWS accounts as a DLQ**. b. Enter the Amazon Resource Name (ARN) of the Amazon SQS queue.	

d. To use a customer managed key to encrypt your target input, under **Encryption**, choose **Customize encryption settings (advanced)**.

If you choose this option, enter an existing KMS key ARN or choose **Create an AWS KMS key** to navigate to the AWS KMS console. For more information about how EventBridge Scheduler encrypts your data at rest, see Encryption at rest in the *Amazon EventBridge Scheduler User Guide*.

e. For **Permissions**, choose **Use existing role**, then select the role.

To have EventBridge Scheduler create a new execution role for you, choose **Create new role for this schedule**. Then, enter a name for **Role name**. If you choose this option, EventBridge Scheduler attaches the required permissions necessary for your templated target to the role.

10. Choose **Next**.
11. In the **Review and create schedule** page, review the details of your schedule. In each section, choose **Edit** to go back to that step and edit its details.
12. Choose **Create schedule**.

    You can view a list of your new and existing schedules on the **Schedules** page. Under the **Status** column, verify that your new schedule is **Enabled**.

Next steps

You can use the EventBridge Scheduler console or the AWS CLI to manage the schedule. For more information, see Managing a schedule in the *Amazon EventBridge Scheduler User Guide*.

Stopping an Amazon ECS task

If you no longer need to keep a standalone task running, you can stop the task. The Amazon ECS console makes it easy to stop one or more tasks.

You can't restart a standalone stopped task.

If you want to stop a service, see Deleting an Amazon ECS service using the console.

To stop a standalone task (AWS Management Console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster to navigate to the cluster details page.

4. On the cluster detail page, choose the **Tasks** tab.

5. You can filter tasks by launch type using the **Filter launch type** list.

Tasks to stop	Steps	
One or more	a. Select the tasks, and then choose **Stop**, **Stop selected**. b. On the **Stop task confirmation page**, choose **Stop**	
All	**Important** If you choose to stop all tasks using the console, Amazon ECS stops all standalone tasks and tasks that are part of a service. Therefore , we recommend caution when using this option. a. Choose **Stop**, **Stop all**. b. On the **Stop task confirmation page**, enter **Stop all tasks**, and then choose **Stop**.	

Amazon ECS services

You can use an Amazon ECS service to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. If one of your tasks fails or stops, the Amazon ECS service scheduler launches another instance of your task definition to replace it. This helps maintain your desired number of tasks in the service.

You can also optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

We recommend that you use the service scheduler for long running stateless services and applications. The service scheduler ensures that the scheduling strategy that you specify is followed and reschedules tasks when a task fails. For example, if the underlying infrastructure fails, the service scheduler reschedules a task. You can use task placement strategies and constraints to customize how the scheduler places and terminates tasks. If a task in a service stops, the scheduler launches a new task to replace it. This process continues until your service reaches your desired number of tasks based on the scheduling strategy that the service uses. The scheduling strategy of the service is also referred to as the *service type*.

The service scheduler also replaces tasks determined to be unhealthy after a container health check or a load balancer target group health check fails. This replacement depends on the `maximumPercent` and `desiredCount` service definition parameters. If a task is marked unhealthy, the service scheduler will first start a replacement task. Then, the following happens.

- If the replacement task has a health status of `HEALTHY`, the service scheduler stops the unhealthy task
- If the replacement task has a health status of `UNHEALTHY`, the scheduler will stop either the unhealthy replacement task or the existing unhealthy task to get the total task count to equal `desiredCount`.

If the `maximumPercent` parameter limits the scheduler from starting a replacement task first, the scheduler will stop an unhealthy task one at a time at random to free up capacity, and then start a replacement task. The start and stop process continues until all unhealthy tasks are replaced with healthy tasks. Once all unhealthy tasks have been replaced and only healthy tasks are running, if the total task count exceeds the `desiredCount`, healthy tasks are stopped at random until the total task count equals `desiredCount`. For more information about `maximumPercent` and `desiredCount`, see [Service definition parameters](#).

The service scheduler includes logic that throttles how often tasks are restarted if tasks repeatedly fail to start. If a task is stopped without having entered a `RUNNING` state, the service scheduler starts to slow down the launch attempts and sends out a service event message. This behavior prevents unnecessary resources from being used for failed tasks before you can resolve the issue. After the service is updated, the service scheduler resumes normal scheduling behavior. For more information, see Amazon ECS service throttle logic and Viewing Amazon ECS service event messages.

Infrastructure compute option

There are two compute options that distribute your tasks.

- A capacity provider strategy causes Amazon ECS to distribute your tasks in one or across multiple capacity providers.

  For the **capacity provider strategy**, the console selects a compute option by default. The following describes the order that the console uses to select a default:

  - If your cluster has a default capacity provider strategy defined, it is selected.
  - If your cluster doesn't have a default capacity provider strategy defined but you have the Fargate capacity providers added to the cluster, a custom capacity provider strategy that uses the `FARGATE` capacity provider is selected.
  - If your cluster doesn't have a default capacity provider strategy defined but you have one or more Auto Scaling group capacity providers added to the cluster, the **Use custom (Advanced)** option is selected and you need to manually define the strategy.
  - If your cluster doesn't have a default capacity provider strategy defined and no capacity providers added to the cluster, the Fargate launch type is selected.
- A launch type causes Amazon ECS to launch our tasks directly on either Fargate or on the EC2 instances registered to your clusters.

  By default the service starts in the subnets in your cluster VPC.

Service auto scaling

Service auto scaling is the ability to increase or decrease the desired number of tasks in your Amazon ECS service automatically. Amazon ECS leverages the Application Auto Scaling service to provide this functionality.

For more information, see Automatically scale your Amazon ECS service.

Service load balancing

Amazon ECS services hosted on AWS Fargate support the Application Load Balancers, Network Load Balancers, and Gateway Load Balancers. Use the following table to learn about what type of load balancer to use.

Load Balancer type	Use in these cases	
Application Load Balancer	Route HTTP/HTTPS (or layer 7) traffic. Application Load Balancers offer several features that make them attractive for use with Amazon ECS services: • Each service can serve traffic from multiple load balancers and expose multiple load balanced ports by specifying multiple target groups. • They are supported by tasks hosted on both Fargate and EC2 instances. • Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance). • Application Load Balancers support path-based routing and priority rules (so that	

Load Balancer type	Use in these cases	
	multiple services can use the same listener port on a single Application Load Balancer).	
Network Load Balancer	Route TCP or UDP (or layer 4) traffic.	
Gateway Load Balancer	Route TCP or UDP (or layer 4) traffic. Use virtual appliances, such as firewalls, intrusion detection and prevention systems, and deep packet inspection systems.	

For more information, see Use load balancing to distribute Amazon ECS service traffic.

Interconecting services

If you need an application to connect to other applications that run as Amazon ECS services, Amazon ECS provides the following ways to do this without a load balancer:

- Service Connect - Allows for service-to-service communications with automatic discovery using short names and standard ports.
- Service discovery - Service discovery uses Route 53 to create a namespace for your service, which allows it to be discoverable through DNS.
- Amazon VPC Lattice - VPC Lattice is a fully managed application networking service to connect, secure, and monitor your services across multiple accounts and VPCs. There is a cost associated with it.

For more information, see Interconnect Amazon ECS services.

Amazon ECS service deployment controllers and strategies

Before you deploy your service, determine the options for deploying it, and the features the service uses.

Scheduling strategy

There are two service scheduler strategies available:

- `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica scheduling strategy.
- `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon scheduling strategy.

> **Note**
>
> Fargate tasks do not support the `DAEMON` scheduling strategy.

Replica scheduling strategy

The *replica* scheduling strategy places and maintains the desired number of tasks in your cluster.

For a service that runs tasks on Fargate, when the service scheduler launches new tasks or stops running tasks, the service scheduler uses a best attempt to maintain a balance across Availability Zones. You don't need to specify task placement strategies or constraints.

When you create a service that runs tasks on EC2 instances, you can optionally specify task placement strategies and constraints to customize task placement decisions. If no task placement strategies or constraints are specified, then by default the service scheduler spreads the tasks across Availability Zones. The service scheduler uses the following logic:

- Determines which of the container instances in your cluster can support your service's task definition (for example, required CPU, memory, ports, and container instance attributes).

- Determines which container instances satisfy any placement constraints that are defined for the service.
- When you have a replica service that depends on a daemon service (for example, a daemon log router task that needs to be running before tasks can use logging), create a task placement constraint that ensures that the daemon service tasks get placed on the EC2 instance prior to the replica service tasks. For more information, see Example Amazon ECS task placement constraints.
- When there's a defined placement strategy, use that strategy to select an instance from the remaining candidates.
- When there's no defined placement strategy, use the following logic to balance tasks across the Availability Zones in your cluster:
  - Sorts the valid container instances. Gives priority to instances that have the fewest number of running tasks for this service in their respective Availability Zone. For example, if zone A has one running service task and zones B and C each have zero, valid container instances in either zone B or C are considered optimal for placement.
  - Places the new service task on a valid container instance in an optimal Availability Zone based on the previous steps. Favors container instances with the fewest number of running tasks for this service.

We recommend that you use the service rebalancing feature when you use the `REPLICA` strategy because it helps ensure high availability for your service.

Daemon scheduling strategy

The *daemon* scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints specified in your cluster. The service scheduler evaluates the task placement constraints for running tasks, and stops tasks that don't meet the placement constraints. When you use this strategy, you don't need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies.

Amazon ECS reserves container instance compute resources including CPU, memory, and network interfaces for the daemon tasks. When you launch a daemon service on a cluster with other replica services, Amazon ECS prioritizes the daemon task. This means that the daemon task is the first task to launch on the instances and the last task to stop after all replica tasks are stopped. This strategy ensures that resources aren't used by pending replica tasks and are available for the daemon tasks.

The daemon service scheduler doesn't place any tasks on instances that have a `DRAINING` status. If a container instance transitions to a `DRAINING` status, the daemon tasks on it are stopped. The service scheduler also monitors when new container instances are added to your cluster and adds the daemon tasks to them.

When you specify a deployment configuration, the value for the `maximumPercent` parameter must be `100` (specified as a percentage), which is the default value used if not set. The default value for the `minimumHealthyPercent` parameter is `0` (specified as a percentage).

You must restart the service when you change the placement constraints for the daemon service. Amazon ECS dynamically updates the resources that are reserved on qualifying instances for the daemon task. For existing instances, the scheduler tries to place the task on the instance.

A new deployment starts when there is a change to the task size or container resource reservation in the task definition. A new deployment also starts when updating a service or setting a different revision of the task definition. Amazon ECS picks up the updated CPU and memory reservations for the daemon, and then blocks that capacity for the daemon task.

If there are insufficient resources for either of the above cases, the following happens:

- The task placement fails.
- A CloudWatch event is generated.
- Amazon ECS continues to try and schedule the task on the instance by waiting for resources to become available.
- Amazon ECS frees up any reserved instances that no longer meet the placement constraint criteria and stops the corresponding daemon tasks.

The daemon scheduling strategy can be used in the following cases:

- Running application containers
- Running support containers for logging, monitoring and tracing tasks

Tasks using the Fargate launch type or the `CODE_DEPLOY` or `EXTERNAL` deployment controller types don't support the daemon scheduling strategy.

When the service scheduler stops running tasks, it attempts to maintain balance across the Availability Zones in your cluster. The scheduler uses the following logic:

- If a placement strategy is defined, use that strategy to select which tasks to terminate. For example, if a service has an Availability Zone spread strategy defined, a task is selected that leaves the remaining tasks with the best spread.
- If no placement strategy is defined, use the following logic to maintain balance across the Availability Zones in your cluster:
  - Sort the valid container instances. Give priority to instances that have the largest number of running tasks for this service in their respective Availability Zone. For example, if zone A has one running service task and zones B and C each have two running service task, container instances in either zone B or C are considered optimal for termination.
  - Stop the task on a container instance in an optimal Availability Zone based on the previous steps. Favoring container instances with the largest number of running tasks for this service.

Deployment controllers

The deployment controller is the mechanism that determines how tasks are deployed for your service. The valid options are:

- ECS

  When you create a service which uses the `ECS` deployment controller, you can choose between the following deployment strategies:

  - `ROLLING`: When you create a service which uses the *rolling update* (`ROLLING`) deployment strategy, the Amazon ECS service scheduler replaces the currently running tasks with new tasks. The number of tasks that Amazon ECS adds or removes from the service during a rolling update is controlled by the service deployment configuration.

    Rolling update deployments are best suited for the following scenarios:

    - Gradual service updates: You need to update your service incrementally without taking the entire service offline at once.
    - Limited resource requirements: You want to avoid the additional resource costs of running two complete environments simultaneously (as required by blue/green deployments).
    - Acceptable deployment time: Your application can tolerate a longer deployment process, as rolling updates replace tasks one by one.
    - No need for instant roll back: Your service can tolerate a rollback process that takes minutes rather than seconds.

  - Simple deployment process: You prefer a straightforward deployment approach without the complexity of managing multiple environments, target groups, and listeners.
  - No load balancer requirement: Your service doesn't use or require a load balancer, Application Load Balancer, Network Load Balancer, or Service Connect (which are required for blue/green deployments).
  - Stateful applications: Your application maintains state that makes it difficult to run two parallel environments.
  - Cost sensitivity: You want to minimize deployment costs by not running duplicate environments during deployment.

  Rolling updates are the default deployment strategy for services and provide a balance between deployment safety and resource efficiency for many common application scenarios.

- `BLUE_GREEN`: A *blue/green* deployment strategy (`BLUE_GREEN`) is a release methodology that reduces downtime and risk by running two identical production environments called blue and green. With Amazon ECS blue/green deployments, you can validate new service revisions before directing production traffic to them. This approach provides a safer way to deploy changes with the ability to quickly roll back if needed.

  Amazon ECS blue/green deployments are best suited for the following scenarios:

  - Service validation: When you need to validate new service revisions before directing production traffic to them
  - Zero downtime: When your service requires zero-downtime deployments
  - Instant roll back: When you need the ability to quickly roll back if issues are detected
  - Load balancer requirement: When your service uses Application Load Balancer, Network Load Balancer, or Service Connect

- External

  Use a third-party deployment controller.

- Blue/green deployment (powered by AWS CodeDeploy)

  CodeDeploy installs an updated version of the application as a new replacement task set and reroutes production traffic from the original application task set to the replacement task set. The original task set is terminated after a successful deployment. Use this deployment controller to verify a new deployment of a service before sending production traffic to it.

Amazon ECS deployment failure detection

Amazon ECS provides two methods for detecting deployment failures:

- Deployment Circuit Breaker
- CloudWatch Alarms

Both methods can be configured to automatically roll back failed deployments to the last known good state.

Consider the following:

- Both methods only support rolling update deployment and blue/green deployment types.
- When both methods are used, either can trigger deployment failure.
- The rollback method requires a previous deployment in COMPLETED state.
- EventBridge events are generated for deployment state changes.

How the Amazon ECS deployment circuit breaker detects failures

The deployment circuit breaker is the rolling update mechanism that determines if the tasks reach a steady state. The deployment circuit breaker has an option that will automatically roll back a failed deployment to the deployment that is in the `COMPLETED` state.

When a service deployment changes state, Amazon ECS sends a service deployment state change event to EventBridge. This provides a programmatic way to monitor the status of your service deployments. For more information, see [Amazon ECS service deployment state change events](). We recommend that you create and monitor an EventBridge rule with an `eventName` of `SERVICE_DEPLOYMENT_FAILED` so that you can take manual action to start your deployment. For more information, see [Getting started with EventBridge]() in the *Amazon EventBridge User Guide*.

When the deployment circuit breaker determines that a deployment failed, it looks for the most recent deployment that is in a `COMPLETED` state. This is the deployment that it uses as the roll-back deployment. When the rollback starts, the deployment changes from a `COMPLETED` to `IN_PROGRESS`. This means that the deployment is not eligible for another rollback until it reaches a `COMPLETED` state. When the deployment circuit breaker does not find a deployment that is in a `COMPLETED` state, the circuit breaker does not launch new tasks and the deployment is stalled.

When you create a service, the scheduler keeps track of the tasks that failed to launch in two stages.

- Stage 1 - The scheduler monitors the tasks to see if they transition into the RUNNING state.
  - Success - The deployment has a chance of transitioning to the COMPLETED state because there is more than one task that transitioned to the RUNNING state. The failure criteria is skipped and the circuit breaker moves to stage 2.
  - Failure - There are consecutive tasks that did not transition to the RUNNING state and the deployment might transition to the FAILED state.
- Stage 2 - The deployment enters this stage when there is at least one task in the RUNNING state. The circuit breaker checks the health checks for the tasks in the current deployment being evaluated. The validated health checks are Elastic Load Balancing, AWS Cloud Map service health checks, and container health checks.
  - Success - There is at least one task in the running state with health checks that have passed.
  - Failure - The tasks that are replaced because of health check failures have reached the failure threshold.

Consider the following when you use the deployment circuit breaker method on a service. EventBridge generates the rule.

- The `DescribeServices` response provides insight into the state of a deployment, the `rolloutState` and `rolloutStateReason`. When a new deployment is started, the rollout state begins in an `IN_PROGRESS` state. When the service reaches a steady state, the rollout state transitions to `COMPLETED`. If the service fails to reach a steady state and circuit breaker is turned on, the deployment will transition to a `FAILED` state. A deployment in a `FAILED` state doesn't launch any new tasks.
- In addition to the service deployment state change events Amazon ECS sends for deployments that have started and have completed, Amazon ECS also sends an event when a deployment with circuit breaker turned on fails. These events provide details about why a deployment failed or if a deployment was started because of a rollback. For more information, see [Amazon ECS service deployment state change events](#).
- If a new deployment is started because a previous deployment failed and a rollback occurred, the `reason` field of the service deployment state change event indicates the deployment was started because of a rollback.

- The deployment circuit breaker is only supported for Amazon ECS services that use the rolling update (ECS) deployment controller.
- You must use the Amazon ECS console, or the AWS CLI when you use the deployment circuit breaker with the CloudWatch option. For more information, see the section called "Create a service using defined parameters" and create-service in the *AWS Command Line Interface Reference*.

The following `create-service` AWS CLI example shows how to create a Linux service when the deployment circuit breaker is used with the rollback option.

```
aws ecs create-service \
     --service-name MyService \
     --deployment-controller type=ECS \
     --desired-count 3 \
     --deployment-configuration "deploymentCircuitBreaker={enable=true,rollback=true}"
 \
     --task-definition sample-fargate:1 \
     --launch-type FARGATE \
     --platform-family LINUX \
     --platform-version 1.4.0 \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
```

Example:

Deployment 1 is in a `COMPLETED` state.

Deployment 2 cannot start, so the circuit breaker rolls back to Deployment 1. Deployment 1 transitions to the `IN_PROGRESS` state.

Deployment 3 starts and there is no deployment in the `COMPLETED` state, so Deployment 3 cannot roll back, or launch tasks.

Failure threshold

The deployment circuit breaker calculates the threshold value, and then uses the value to determine when to move the deployment to a `FAILED` state.

The deployment circuit breaker has a minimum threshold of 3 and a maximum threshold of 200. and uses the values in the following formula to determine the deployment failure.

```
Minimum threshold <= 0.5 * desired task count => maximum threshold
```

When the result of the calculation is greater than the minimum of 3, but smaller than the maximum of 200, the failure threshold is set to the calculated threshold (rounded up).

Note

You cannot change either of the threshold values.

There are two stages for the deployment status check.

1. The deployment circuit breaker monitors tasks that are part of the deployment and checks for tasks that are in the `RUNNING` state. The scheduler ignores the failure criteria when a task in the current deployment is in the `RUNNING` state and proceeds to the next stage. When tasks fail to reach in the `RUNNING` state, the deployment circuit breaker increases the failure count by one. When the failure count equals the threshold, the deployment is marked as `FAILED`.
2. This stage is entered when there are one or more tasks in the `RUNNING` state. The deployment circuit breaker performs health checks on the following resources for the tasks in the current deployment:
   - Elastic Load Balancing load balancers
   - AWS Cloud Map service
   - Amazon ECS container health checks

   When a health check fails for the task, the deployment circuit breaker increases the failure count by one. When the failure count equals the threshold, the deployment is marked as `FAILED`.

The following table provides some examples.

Desired task count	Calculation	Threshold
1	`3 <= 0.5 * 1 => 200`	3 (the calculated value is less than the minimum)
25	`3 <= 0.5 * 25 => 200`	13 (the value is rounded up)

Desired task count	Calculation	Threshold
400	`3 <= 0.5 * 400 => 200`	200
800	`3 <= 0.5 * 800 => 200`	200 (the calculated value is greater than the maximum)

For example, when the threshold is 3, the circuit breaker starts with the failure count set at 0. When a task fails to reach the `RUNNING` state, the deployment circuit breaker increases the failure count by one. When the failure count equals 3, the deployment is marked as `FAILED`.

For additional examples about how to use the rollback option, see [Announcing Amazon ECS deployment circuit breaker](#).

How CloudWatch alarms detect Amazon ECS deployment failures

You can configure Amazon ECS to set the deployment to failed when it detects that a specified CloudWatch alarm has gone into the `ALARM` state.

You can optionally set the configuration to roll back a failed deployment to the last completed deployment.

The following `create-service` AWS CLI example shows how to create a Linux service when the deployment alarms are used with the rollback option.

```
aws ecs create-service \
     --service-name MyService \
     --deployment-controller type=ECS \
     --desired-count 3 \
     --deployment-configuration
 "alarms={alarmNames=[alarm1Name,alarm2Name],enable=true,rollback=true}" \
     --task-definition sample-fargate:1 \
     --launch-type FARGATE \
     --platform-family LINUX \
     --platform-version 1.4.0 \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
```

Consider the following when you use the Amazon CloudWatch alarms method on a service.

- The duration when both blue and green service revisions are running simultaneously after the production traffic has shifted. Amazon ECS computes this time period based on the alarm configuration associated with the deployment. You can't set this value.
- The `deploymentConfiguration` request parameter now contains the `alarms` data type. You can specify the alarm names, whether to use the method, and whether to initiate a rollback when the alarms indicate a deployment failure. For more information, see [CreateService](#) in the *Amazon Elastic Container Service API Reference*.
- The `DescribeServices` response provides insight into the state of a deployment, the `rolloutState` and `rolloutStateReason`. When a new deployment starts, the rollout state begins in an `IN_PROGRESS` state. When the service reaches a steady state and the bake time is complete, the rollout state transitions to `COMPLETED`. If the service fails to reach a steady state and the alarm has gone into the `ALARM` state, the deployment will transition to a `FAILED` state. A deployment in a `FAILED` state won't launch any new tasks.
- In addition to the service deployment state change events Amazon ECS sends for deployments that have started and have completed, Amazon ECS also sends an event when a deployment that uses alarms fails. These events provide details about why a deployment failed or if a deployment was started because of a rollback. For more information, see [Amazon ECS service deployment state change events](#).
- If a new deployment is started because a previous deployment failed and rollback was turned on, the `reason` field of the service deployment state change event will indicate the deployment was started because of a rollback.
- If you use the deployment circuit breaker and the Amazon CloudWatch alarms to detect failures, either one can initiate a deployment failure as soon as the criteria for either method is met. A rollback occurs when you use the rollback option for the method that initiated the deployment failure.
- The Amazon CloudWatch alarms is only supported for Amazon ECS services that use the rolling update (`ECS`) deployment controller.
- You can configure this option by using the Amazon ECS console, or the AWS CLI. For more information, see [the section called "Create a service using defined parameters"](#) and [create-service](#) in the *AWS Command Line Interface Reference*.
- You might notice that the deployment status remains `IN_PROGRESS` for a prolonged amount of time. The reason for this is that Amazon ECS does not change the status until it has deleted the active deployment, and this does not happen until after the bake time. Depending on your alarm configuration, the deployment might appear to take several minutes longer than it does when you don't use alarms (even though the new primary task set is scaled up and the old deployment

is scaled down). If you use CloudFormation timeouts, consider increasing the timeouts. For more information, see Creating wait conditions in a template in the *AWS CloudFormation User Guide*.

- Amazon ECS calls `DescribeAlarms` to poll the alarms. The calls to `DescribeAlarms` count toward the CloudWatch service quotas associated with your account. If you have other AWS services that call `DescribeAlarms`, there might be an impact on Amazon ECS to poll the alarms. For example, if another service makes enough `DescribeAlarms` calls to reach the quota, that service is throttled and Amazon ECS is also throttled and unable to poll alarms. If an alarm is generated during the throttling period, Amazon ECS might miss the alarm and the roll back might not occur. There is no other impact on the deployment. For more information on CloudWatch service quotas, see CloudWatch service quotas in the *CloudWatch User Guide*.
- If an alarm is in the `ALARM` state at the beginning of a deployment, Amazon ECS will not monitor alarms for the duration of that deployment (Amazon ECS ignores the alarm configuration). This behavior addresses the case where you want to start a new deployment to fix an initial deployment failure.

Recommended alarms

We recommend that you use the following alarm metrics:

- If you use an Application Load Balancer, use the `HTTPCode_ELB_5XX_Count` and `HTTPCode_ELB_4XX_Count` Application Load Balancer metrics. These metrics check for HTTP spikes. For more information about the Application Load Balancer metrics, see CloudWatch metrics for your Application Load Balancer in the *User Guide for Application Load Balancers*.
- If you have an existing application, use the `CPUUtilization` and `MemoryUtilization` metrics. These metrics check for the percentage of CPU and memory that the cluster or service uses. For more information, see the section called “Considerations”.
- If you use Amazon Simple Queue Service queues in your tasks, use `ApproximateNumberOfMessagesNotVisible` Amazon SQS metric. This metric checks for number of messages in the queue that are delayed and not available for reading immediately. For more information about Amazon SQS metrics, see Available CloudWatch metrics for Amazon SQS in the *Amazon Simple Queue Service Developer Guide*.

Lifecycle hooks for Amazon ECS service deployments

When a deployment starts, it goes through lifecycle stages. These stages can be in states such as IN_PROGRESS or successful. You can use lifecycle hooks, which are Lambda functions that Amazon ECS runs on your behalf at specified lifecycle stages. The functions can be either of the following:

- An asynchronous API which validates the health check within 15 minutes.
- A poll API which initiates another asynchronous process which evaluates the lifecycle hook completion.

After the function has finished running, it must return a `hookStatus` for the deployment to continue. If a `hookStatus` is not returned, or if the function fails, the deployment rolls back. The following are the `hookStatus` values:

- `SUCCEEDED` - the deployment continues to the next lifecycle stage
- `FAILED` – the deployment rolls back to the last successful deployment.
- `IN_PROGRESS` – Amazon ECS runs the function again after a short period of time. By default this is a 30 second interval, however this value is customizable by returning a `callBackDelay` alongside the `hookStatus`.

The following example shows how to return a `hookStatus` with a custom callback delay. In this example, Amazon ECS would retry this hook in 60 seconds instead of the default 30 seconds:

```
{
    "hookStatus": "IN_PROGRESS",
    "callBackDelay": 60
}
```

When a roll back happens, Amazon ECS runs the lifecycle hooks for the following lifecycle stages:

- PRODUCTION_TRAFFIC_SHIFT
- TEST_TRAFFIC_SHIFT

Lifecycle payloads

When you configure lifecycle hooks for your ECS service deployments, Amazon ECS invokes these hooks at specific stages of the deployment process. Each lifecycle stage provides a JSON payload with information about the current state of the deployment. This document describes the payload structure for each lifecycle stage.

Common payload structure

All lifecycle stage payloads include the following common fields:

- `serviceArn` - The Amazon Resource Name (ARN) of the service.
- `targetServiceRevisionArn` - The ARN of the target service revision being deployed.
- `testTrafficWeights` - A map of service revision ARNs to their corresponding test traffic weight percentages.
- `productionTrafficWeights` - A map of service revision ARNs to their corresponding production traffic weight percentages.

Lifecycle stage payloads

RECONCILE_SERVICE

This stage occurs at the beginning of the deployment process when the service is being reconciled. The following shows an example payload for this lifecycle stage.


```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  },
  "productionTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  }
}
```

Expectations at this stage:

- Primary task set is at 0% scale

PRE_SCALE_UP

This stage occurs before the new tasks are scaled up. The following shows an example payload for this lifecycle stage.


```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```


Expectations at this stage:

- The green service revision tasks are at 0% scale

POST_SCALE_UP

This stage occurs after the new tasks have been scaled up and are healthy. The following shows an example payload for this lifecycle stage.


```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```


Expectations at this stage:

- The green service revision tasks are at 100% scale
- Tasks in the green service revision are healthy

TEST_TRAFFIC_SHIFT

This stage occurs when test traffic is being shifted to the green service revision tasks.

The following shows an example payload for this lifecycle stage.


```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  },
  "productionTrafficWeights": {}
}
```


Expectations at this stage:

- Test traffic is in the process of moving towards the green service revision tasks.

POST_TEST_TRAFFIC_SHIFT

This stage occurs after test traffic has been fully shifted to the new tasks.

The following shows an example payload for this lifecycle stage.


```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```

Expectations at this stage:

- 100% of test traffic moved towards the green service revision tasks.

PRODUCTION_TRAFFIC_SHIFT

This stage occurs when production traffic is being shifted to the green service revision tasks.

```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/01275892":
 100,
    "arn:aws:ecs:us-west-2:1234567890:service-revision/myCluster/myService/78652123": 0
  }
}
```

Expectations at this stage:

- Production traffic is in the process of moving to the green service revision.

POST_PRODUCTION_TRAFFIC_SHIFT

This stage occurs after production traffic has been fully shifted to the green service revision tasks.

```
{
  "serviceArn": "arn:aws:ecs:us-west-2:1234567890:service/myCluster/myService",
  "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:1234567890:service-revision/
myCluster/myService/01275892",
  "testTrafficWeights": {},
  "productionTrafficWeights": {}
}
```

Expectations at this stage:

- 100% of production traffic moved towards the green service revision tasks.

Lifecycle stage categories

Lifecycle stages fall into two categories:

1. **Single invocation stages** - These stages are invoked only once during a service deployment:
   - PRE_SCALE_UP
   - POST_SCALE_UP
   - POST_TEST_TRAFFIC_SHIFT
   - POST_PRODUCTION_TRAFFIC_SHIFT
2. **Recurring invocation stages** - These stages might be invoked multiple times during a service deployment, for example when a rollback operation happens:
   - TEST_TRAFFIC_SHIFT
   - PRODUCTION_TRAFFIC_SHIFT

Deployment status during lifecycle hooks

While lifecycle hooks are running, the deployment status will be `IN_PROGRESS` for all lifecycle stages.

Lifecycle Stage	Deployment Status
RECONCILE_SERVICE	IN_PROGRESS
PRE_SCALE_UP	IN_PROGRESS
POST_SCALE_UP	IN_PROGRESS
TEST_TRAFFIC_SHIFT	IN_PROGRESS
POST_TEST_TRAFFIC_SHIFT	IN_PROGRESS
PRODUCTION_TRAFFIC_SHIFT	IN_PROGRESS
POST_PRODUCTION_TRAFFIC_SHIFT	IN_PROGRESS

Stopping Amazon ECS service deployments

You can manually stop a deployment when a failing deployment was not detected by the circuit breaker or CloudWatch alarms. The following stop types are available:

- Rollback - This option rolls back the service deployment to the previous service revision.

  You can use this option even if you didn't configure the service deployment for the rollback option.

You can stop a deployment that is in any of the following states. For more information about service deployment states, see View service history using Amazon ECS service deployments.

- PENDING - The service deployment moves to the ROLLBACK_REQUESTED state, and then the rollback operation starts.
- IN_PROGRESS - The service deployment moves to the ROLLBACK_REQUESTED state, and then the rollback operation starts.
- STOP_REQUESTED - The service deployment continues to stop.
- ROLLBACK_REQUESTED - The service deployment continues the rollback operation.
- ROLLBACK_IN_PROGRESS - The service deployment continues the rollback operation.

Procedure

Before you begin, configure the required permissions for viewing service deployments. For more information, see Permissions required for viewing Amazon ECS service deployments.

Amazon ECS Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, choose the service.

   The service details page displays.
4. On the service details page, choose **Deployments**.

   The deployments page displays.

5. Under **Ongoing deployment**, choose **Roll back**. Then, in the confirmation window, choose **Roll back**.

AWS CLI

1. Run `list-service-deployments` to retrieve the service deployment ARN.

   Replace the *user-input* with your values.

   ```
   aws ecs list-service-deployments --cluster cluster-name --service service-name
   ```

   Note the `serviceDeploymentArn` for the deployment you want to stop.

   ```
   {
       "serviceDeployments": [
           {
               "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/cluster-name/service-name/NCWGC2ZR-taawPAYrIaU5",
               "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/cluster-name/service-name",
               "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/cluster-name",
               "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/cluster-name/service-name/4980306466373577095",
               "status": "SUCCESSFUL"
           }
       ]
   }
   ```

2. Run `stop-service-deployments`. Use the `serviceDeploymentArn` that was returned from `list-service-deployments`.

   Replace the *user-input* with your values.

   ```
   aws ecs stop-service-deployment --service-deployment-arn
    arn:aws:ecs:region:123456789012:service-deployment/cluster-name/service-name/NCWGC2ZR-taawPAYrIaU5 --stop-type ROLLBACK
   ```

Next steps

Decide what changes need to be made to the service, and then update the service. For more information, see Updating an Amazon ECS service.

Deploy Amazon ECS services by replacing tasks

When you create a service which uses the *rolling update* (`ECS`) deployment type, the Amazon ECS service scheduler replaces the currently running tasks with new tasks. The number of tasks that Amazon ECS adds or removes from the service during a rolling update is controlled by the service deployment configuration.

Amazon ECS uses the following parameters to determine the number of tasks:

- The `minimumHealthyPercent` represents the lower limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for the service. This value is rounded up. For example if the minimum healthy percent is `50` and the desired task count is four, then the scheduler can stop two existing tasks before starting two new tasks. Likewise, if the minimum healthy percent is 75% and the desired task count is two, then the scheduler can't stop any tasks due to the resulting value also being two.

  If tasks become unhealthy, the Amazon ECS service scheduler will start replacement tasks first and maintain `minimumHealthyPercent` tasks until the replacement tasks become healthy. As the replacement tasks launch and become healthy, the unhealthy tasks will gradually be stopped.
- The `maximumPercent` represents the upper limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for a service. This value is rounded down. For example if the maximum percent is `200` and the desired task count is four then the scheduler can start four new tasks before stopping four existing tasks. Likewise, if the maximum percent is `125` and the desired task count is three, the scheduler can't start any tasks due to the resulting value also being three.

> **Important**
>
> When setting a minimum healthy percent or a maximum percent, you should ensure that the scheduler can stop or start at least one task when a deployment is initiated. If your service has a deployment that is stuck due to an invalid deployment configuration, a

service event message will be sent. For more information, see service (*service-name*) was unable to stop or start tasks during a deployment because of the service deployment configuration. Update the minimumHealthyPercent or maximumPercent value and try again..

Rolling deployments have 2 methods which provide a way to quickly identify when a service deployment has failed:

- the section called " How the deployment circuit breaker detects failures"
- the section called "How CloudWatch alarms detect deployment failures"

The methods can be used separately or together. When both methods are used, the deployment is set to failed as soon as the failure criteria for either failure method is met.

Use the following guidelines to help determine which method to use:

- Circuit breaker - Use this method when you want to stop a deployment when the tasks can't start.
- CloudWatch alarms - Use this method when you want to stop a deployment based on application metrics.

Both methods support rolling back to the previous service revision.

Container image resolution

By default, Amazon ECS resolves container image tags specified in the task definition to container image digests. If you create a service that runs and maintains a single task, that task is used to establish image digests for the containers in the task. If you create a service that runs and maintains multiple tasks, the first task started by the service scheduler during deployment is used to establish the image digests for the containers in the tasks.

If three or more attempts at establishing the container image digests fail, the deployment continues without image digest resolution. If the deployment circuit breaker is enabled, the deployment is additionally failed and rolled back.

After the container image digests have been established, Amazon ECS uses the digests to start any other desired tasks, and for any future service updates. This leads to all tasks in a service always running identical container images, resulting in version consistency for your software.

You can configure this behavior for each container in your task by using the `versionConsistency` parameter in the container definition. For more information, see [`versionConsistency`].

Note

- Amazon ECS Agent versions lower than `1.31.0` don't support image digest resolution. Agent versions `1.31.0` to `1.69.0` support image digest resolution only for images pushed to Amazon ECR repositories. Agent versions `1.70.0` or higher support image digest resolution for all images.
- The minimum Fargate Linux platform version for image digest resolution is `1.3.0`. The minimum Fargate Windows platform version for image digest resolution is `1.0.0`.
- Amazon ECS doesn't capture digests of sidecar containers managed by Amazon ECS, such as the Amazon GuardDuty security agent or Service Connect proxy.
- To reduce potential latency associated with container image resolution in services with multiple tasks, run Amazon ECS agent version `1.83.0` or higher on EC2 container instances. To avoid potential latency, specify container image digests in your task definition.
- If you create a service with a desired task count of zero, Amazon ECS can't establish container digests until you trigger another deployment of the service with a desired task count greater than zero.
- To establish updated image digests, you can force a new deployment. The updated digests will be used to start new tasks and will not affect already running tasks. For more information about forcing new deployments, see forceNewDeployment in the *Amazon ECS API reference*.
- When using EC2 capacity providers, if there is insufficient capacity to start a task during the initial deployment, software version consistency may fail. To ensure version consistency is maintained even when capacity is limited, explicitly set `versionConsistency: "enabled"` in your task definition container configuration rather than relying on the default behavior. This causes Amazon ECS to wait until capacity becomes available before proceeding with the deployment.

Best practices for Amazon ECS service parameters

To ensure that there's no application downtime, the deployment process is as follows:

1. Start the new application containers while keeping the existing containers running.
2. Check that the new containers are healthy.
3. Stop the old containers.

Depending on your deployment configuration and the amount of free, unreserved space in your cluster it may take multiple rounds of this to complete replace all old tasks with new tasks.

There are two service configuration options that you can use to modify the number:

- `minimumHealthyPercent`: 100% (default)

  The lower limit on the number of tasks for your service that must remain in the `RUNNING` state during a deployment. This is a percentage of the `desiredCount` rounded up to the nearest integer. This parameter allows you to deploy without using additional cluster capacity.
- `maximumPercent`: 200% (default)

  The upper limit on the number of tasks for your service that are allowed in the `RUNNING` or `PENDING` state during a deployment. This is a percentage of the `desiredCount` rounded down to the nearest integer.

Example: Default configuration options

Consider the following service that has six tasks, deployed in a cluster that has room for eight tasks total. The default service configuration options don't allow the deployment to go below 100% of the six desired tasks.

The deployment process is as follows:

1. The goal is to replace the six tasks.
2. The scheduler starts two new tasks because the default settings require that there are six running tasks.

   There are now six existing tasks and two new tasks.
3. The scheduler stops two of the existing tasks.

There are now four existing tasks and two new ones.

4. The scheduler starts two additional new tasks.

   There are now four existing tasks and four new tasks.

5. The scheduler shuts down two of the existing tasks.

   There are now two existing tasks and four new ones.

6. The scheduler starts two additional new tasks.

   There are now two existing tasks and six new tasks

7. The scheduler shuts down the last two existing tasks.

   There are now six new tasks.

In the above example, if you use the default values for the options, there is a 2.5 minute wait for each new task that starts. Additionally, the load balancer might have to wait 5 minutes for the old task to stop.

Example: Modify `minimumHealthyPercent`

You can speed up the deployment by setting the `minimumHealthyPercent` value to 50%.

Consider the following service that has six tasks, deployed in a cluster that has room for eight tasks total. The deployment process is as follows:

1. The goal is to replace six tasks.
2. The scheduler stops three of the existing tasks.

   There are still three existing tasks running which meets the `minimumHealthyPercent` value.

3. The scheduler starts five new tasks.

   There are three existing task tasks and five new tasks.

4. The scheduler stops the remaining three existing tasks.

   There are five new tasks

5. The scheduler starts the final new tasks.

   There are six new tasks.

Example: Modify cluster free space

You could also add additional free space so that you can run additional tasks.

Consider the following service that has six tasks, deployed in a cluster that has room for ten tasks total. The deployment process is as follows:

1. The goal is to replace the existing tasks.
2. The scheduler stops three of the existing tasks,

   There are three existing tasks.
3. The scheduler starts six new tasks.

   There are the existing tasks and six new tasks
4. The scheduler stops the three existing tasks.

   There are six new tasks.

Recommendations

Use the following values for the service configuration options when your tasks are idle for some time and don't have a high utilization rate.

- `minimumHealthyPercent`: 50%
- `maximumPercent`: 200%

Creating an Amazon ECS rolling update deployment

Create a service to run and maintain a specified number of instances of a task definition simultaneously in a cluster. If one of your tasks fails or stops, the Amazon ECS service scheduler launches another instance of your task definition to replace it. This helps maintain your desired number of tasks in the service.

Decide on the following configuration parameters before you create a service:

- There are two compute options that distribute your tasks.
  - A **capacity provider strategy** causes Amazon ECS to distribute your tasks in one or across multiple capacity providers.

  - A **launch type** causes Amazon ECS to launch our tasks directly on either Fargate or on the EC2 instances registered to your clusters.
- Task definitions that use the `awsvpc` network mode or services configured to use a load balancer must have a networking configuration. By default, the console selects the default Amazon VPC along with all subnets and the default security group within the default Amazon VPC.
- The placement strategy, The default task placement strategy distributes tasks evenly across Availability Zones.

  We recommend that you use Availability Zone rebalancing to help ensure high availability for your service. For more information, see Balancing an Amazon ECS service across Availability Zones.
- When you use the **Launch Type** for your service deployment, by default the service starts in the subnets in your cluster VPC.
- For the **capacity provider strategy**, the console selects a compute option by default. The following describes the order that the console uses to select a default:
  - If your cluster has a default capacity provider strategy defined, it is selected.
  - If your cluster doesn't have a default capacity provider strategy defined but you have the Fargate capacity providers added to the cluster, a custom capacity provider strategy that uses the `FARGATE` capacity provider is selected.
  - If your cluster doesn't have a default capacity provider strategy defined but you have one or more Auto Scaling group capacity providers added to the cluster, the **Use custom (Advanced)** option is selected and you need to manually define the strategy.
  - If your cluster doesn't have a default capacity provider strategy defined and no capacity providers added to the cluster, the Fargate launch type is selected.
- The default deployment failure detection default options are to use the **Amazon ECS deployment circuit breaker** option with the **Rollback on failures** option.

  For more information, see How the Amazon ECS deployment circuit breaker detects failures.
- Decide if you want Amazon ECS to increase or decrease the desired number of tasks in your service automatically. For information see, Automatically scale your Amazon ECS service.
- If you need an application to connect to other applications that run in Amazon ECS, determine the option that fits your architecture. For more information, see Interconnect Amazon ECS services.
- When you create a service that uses Amazon ECS circuit breaker, Amazon ECS creates a service deployment and a service revision. These resources allow you to view detailed information about

the service history. For more information, see View service history using Amazon ECS service deployments.

For information about how to create a service using the AWS CLI, see create-service in the *AWS Command Line Interface Reference*.

For information about how to create a service using AWS CloudFormation, see AWS::ECS::Service in the *AWS CloudFormation User Guide*.

Create a service with the default options

You can use the console to quickly create and deploy a service. The service has the following configuration:

- Deploys in the VPC and subnets associated with your cluster
- Deploys one task
- Uses the rolling deployment
- Uses the capacity provider strategy with your default capacity provider
- Uses the deployment circuit breaker to detect failures and sets the option to automatically roll back the deployment on failure

To deploy a service using the default parameters follow these steps.

To create a service (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation page, choose **Clusters**.
3. On the **Clusters** page, choose the cluster to create the service in.
4. From the **Services** tab, choose **Create**.

   The **Create service** page appears.
5. Under **Service details**, do the following:

   a. For **Task definition**, enter the task definition family and revision to use.

   b. For **Service name**, enter a name for your service.
6. To use ECS Exec to debug the service, under **Troubleshooting configuration**, select **Turn on ECS Exec**.

7. Under **Deployment configuration**, do the following:

   - For **Desired tasks**, enter the number of tasks to launch and maintain in the service.

8. (Optional) To help identify your service and tasks, expand the **Tags** section, and then configure your tags.

   To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the task definition tags, select **Turn on Amazon ECS managed tags**, and then select **Task definitions**.

   To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the service tags, select **Turn on Amazon ECS managed tags**, and then select **Service**.

   Add or remove a tag.

   - [Add a tag] Choose **Add tag**, and then do the following:
     - For **Key**, enter the key name.
     - For **Value**, enter the key value.
   - [Remove a tag] Next to the tag, choose **Remove tag**.

Create a service using defined parameters

To create a service by using defined parameters, follow these steps.

To create a service (Amazon ECS console)

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. Determine the resource from where you launch the service.

To start a service from	Steps	
Clusters	a. On the **Clusters** page, select the cluster to create the service in. b. From the **Services** tab, choose **Create**.	

To start a service from	Steps	
Task definition	a. On the **Task definitions** page, select the option button next to the task definition. b. On the **Deploy** menu, choose **Create service**.	

The **Create service** page appears.

3. Under Service details, do the following:

   a. For **Task definition**, enter the task definition to use. Then, for **Revision**, choose the revision to use.

   b. For **Service name**, enter a name for your service.

4. For **Existing cluster**, choose the cluster.

   Choose **Create cluster** to run the task on a new cluster

5. Choose how your tasks are distributed across your cluster infrastructure. Under **Compute configuration**, choose your option.

Compute option	Steps	
Capacity provider strategy	a. Under **Compute options**, choose **Capacity provider strategy**. b. Choose a strategy: • To use the cluster's default capacity provider strategy, choose **Use cluster default**. • If your cluster doesn't have a default capacity provider strategy, or to	

Compute option	Steps	
	use a custom strategy, choose **Use custom**, **Add capacity provider strategy**, and then define your custom capacity provider strategy by specifying a **Base**, **Capacity provider**, and **Weight**. **Note** To use a capacity provider in a strategy, the capacity provider must be associated with the cluster.	
Launch type	a. In the **Compute options** section, select **Launch type**. b. For **Launch type**, choose a launch type. c. (Optional) When the Fargate launch type is specified, for **Platform version**, specify the platform version to use. If a platform version isn't specified, the `LATEST` platform version is used.	

6. To use ECS Exec to debug the service, under **Troubleshooting configuration**, select **Turn on ECS Exec**.
7. Under **Deployment configuration**, do the following:

   a. For **Service type**, choose the service scheduling strategy.

      - To have the scheduler deploy exactly one task on each active container instance that meets all of the task placement constraints, choose **Daemon**.
      - To have the scheduler place and maintain the desired number of tasks in your cluster, choose **Replica**.

   b. If you chose **Replica**, for **Desired tasks**, enter the number of tasks to launch and maintain in the service.

   c. If you chose **Replica**, to have Amazon ECS monitor the distribution of tasks across Availability Zones, and redistribute them when there is an imbalance, under **Availability Zone service rebalancing**, select **Availability Zone service rebalancing**.

   d. For **Health check grace period**, enter the amount of time (in seconds) that the enter the amount of time (in seconds) that the service scheduler ignores unhealthy Elastic Load Balancing, VPC Lattice, and container health checks after a task has first started. If you do not specify a health check grace period value, the default value of 0 is used.

   e. Determine the deployment type for your service. Expand **Deployment options**, and then specify the following parameters.

Deployment type	Steps	
Rolling update	a. For **Min running tasks**, enter the lower limit on the number of tasks in the service that must remain in the `RUNNING` state during a deploymen t, as a percentage of the desired number of tasks (rounded up to the nearest integer). For more information, see	

Deployment type	Steps	
	[Deployment configuration](#). b. For **Max running tasks**, enter the upper limit on the number of tasks in the service that are allowed in the RUNNING or PENDING state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer).	

f. To configure how Amazon ECS detects and handles deployment failures, expand **Deployment failure detection**, and then choose your options.

   i. To stop a deployment when the tasks cannot start, select **Use the Amazon ECS deployment circuit breaker**.

      To have the software automatically roll back the deployment to the last completed deployment state when the deployment circuit breaker sets the deployment to a failed state, select **Rollback on failures**.

   ii. To stop a deployment based on application metrics, select **Use CloudWatch alarm(s)**. Then, from **CloudWatch alarm name**, choose the alarms. To create a new alarm, go to the CloudWatch console.

      To have the software automatically roll back the deployment to the last completed deployment state when a CloudWatch alarm sets the deployment to a failed state, select **Rollback on failures**.

8. If your task definition uses the `awsvpc` network mode, you can specify a custom network configuration expand **Networking**, and then do the following:

   a. For **VPC**, select the VPC to use.

   b. For **Subnets**, select one or more subnets in the VPC that the task scheduler considers when placing your tasks.

c. For **Security group**, you can either select an existing security group or create a new one. To use an existing security group, select the security group and move to the next step. To create a new security group, choose **Create a new security group**. You must specify a security group name, description, and then add one or more inbound rules for the security group.

d. For **Public IP**, choose whether to auto-assign a public IP address to the elastic network interface (ENI) of the task.

   AWS Fargate tasks can be assigned a public IP address when run in a public subnet so they have a route to the internet. EC2 tasks can't be assigned a public IP using this field. For more information, see Amazon ECS task networking options for the Fargate launch type and Allocate a network interface for an Amazon ECS task.

9. (Optional) To interconnect your service using Service Connect, expand **Service Connect**, and then specify the following:

   a. Select **Turn on Service Connect**.

   b. Under **Service Connect configuration**, specify the client mode.

      - If your service runs a network client application that only needs to connect to other services in the namespace, choose **Client side only**.
      - If your service runs a network or web service application and needs to provide endpoints for this service, and connects to other services in the namespace, choose **Client and server**.

   c. To use a namespace that is not the default cluster namespace, for **Namespace**, choose the service namespace.

   d. (Optional) Specify a log configuration. Select **Use log collection**. The default option sends container logs to CloudWatch Logs. The other log driver options are configured using AWS FireLens. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.

      The following describes each container log destination in more detail.

      - **Amazon CloudWatch** – Configure the task to send container logs to CloudWatch Logs. The default log driver options are provided, which create a CloudWatch log group on your behalf. To specify a different log group name, change the driver option values.

- **Amazon Data Firehose –** Configure the task to send container logs to Firehose. The default log driver options are provided, which send logs to a Firehose delivery stream. To specify a different delivery stream name, change the driver option values.
- **Amazon Kinesis Data Streams –** Configure the task to send container logs to Kinesis Data Streams. The default log driver options are provided, which send logs to an Kinesis Data Streams stream. To specify a different stream name, change the driver option values.
- **Amazon OpenSearch Service –** Configure the task to send container logs to an OpenSearch Service domain. The log driver options must be provided.
- **Amazon S3 –** Configure the task to send container logs to an Amazon S3 bucket. The default log driver options are provided, but you must specify a valid Amazon S3 bucket name.

10. (Optional) To interconnect your service using Service Discovery, expand **Service discovery**, and then do the following.

    a. Select **Use service discovery**.

    b. To use a new namespace, choose **Create a new namespace** under **Configure namespace**, and then provide a namespace name and description. To use an existing namespace, choose **Select an existing namespace** and then choose the namespace that you want to use.

    c. Provide Service Discovery service information such as the service's name and description.

    d. To have Amazon ECS perform periodic container-level health checks, select **Enable Amazon ECS task health propagation**.

    e. For **DNS record type**, select the DNS record type to create for your service. Amazon ECS service discovery only supports **A** and **SRV** records, depending on the network mode that your task definition specifies. For more information about these record types, see [Supported DNS Record Types](#) in the *Amazon Route 53 Developer Guide*.

    - If the task definition that your service task specifies uses the `bridge` or `host` network mode, only type **SRV** records are supported. Choose a container name and port combination to associate with the record.
    - If the task definition that your service task specifies uses the `awsvpc` network mode, select either the **A** or **SRV** record type. If you choose **A**, skip to the next step. If you choose **SRV**, specify either the port that the service can be found on or a container name and port combination to associate with the record.

For **TTL**, enter the time in seconds how long a record set is cached by DNS resolvers and by web browsers.

11. (Optional) To interconnect your service using VPC Lattice, xxpand **VPC Lattice**, and then do the following:

    a. Select **Turn on VPC Lattice**

    b. For **Infrastructure role**, choose the infrastructure role.

       If you haven't created a role, choose **Create infrastructure role**.

    c. Under **Target Groups** choose the target group or groups. You need to choose at least one target group and can have a maximum of five. Choose **Add target group** to add additional target groups. Choose the **Port name**, **Protocol**, and **Port** for each target group you chose.

       To delete a target group, choose **Remove**.

       Note

       - If you want to add existing target groups, you need use the AWS CLI. For instructions on how to add target groups using the AWS CLI, see register-targets in the *AWS Command Line Interface Reference*.
       - While a VPC Lattice service can have multiple target groups, each target group can only be added to one service.

    d. To complete the VPC Lattice configuration, by including your new target groups in the listener default action or in the rules of an existing VPC Lattice service in the VPC Lattice console. For more information, see Listener rules for your VPC Lattice service.

12. (Optional) To configure a load balancer for your service, expand **Load balancing**.

    Choose the load balancer.

To use this load balancer	Do this	
Application Load Balancer	a. For **Load balancer type**, select **Application Load Balancer**.	

To use this load balancer	Do this	
	b. Choose **Create a new load balancer** to create a new Application Load Balancer or **Use an existing load balancer** to select an existing Application Load Balancer. c. For **Load balancer name**, enter a unique name. d. For **Choose container to load balance**, choose the container that hosts the service. e. For **Listener**, enter a port and protocol for the Application Load Balancer to listen for connection requests on. By default, the load balancer will be configured to use port 80 and HTTP. f. For **Target group name**, enter a name and a protocol for the target group that the Application Load Balancer routes requests to. By default, the target group routes requests to the first container defined in your task definition. g. For **Degregistration delay**, enter the number	

To use this load balancer	Do this	
	of seconds for the load balancer to change the target state to `UNUSED`. The default is 300 seconds. h. For **Health check path**, enter an existing path within your container where the Application Load Balancer periodically sends requests to verify the connection health between the Application Load Balancer and the container. The default is the root directory (/).	

To use this load balancer	Do this	
Network Load Balancer	a. For **Load balancer type**, select **Network Load Balancer**. b. For **Load Balancer**, choose an existing Network Load Balancer. c. For **Choose container to load balance**, choose the container that hosts the service. d. For **Target group name**, enter a name and a protocol for the target group that the Network Load Balancer routes requests to. By default, the target group routes requests to the first container defined in your task definition. e. For **Degregistration delay**, enter the number of seconds for the load balancer to change the target state to `UNUSED`. The default is 300 seconds. f. For **Health check path**, enter an existing path within your container where the Network Load Balancer periodically sends requests to verify	

To use this load balancer	Do this
	the connection health between the Application Load Balancer and the container. The default is the root directory (/).

13. (Optional) To configure service Auto Scaling, expand **Service auto scaling**, and then specify the following parameters.To use predicte auto scaling, which looks at past load data from traffic flows, configure it after you create the service. For more information, see [Use historical patterns to scale Amazon ECS services with predictive scaling](https://docs.aws.amazon.com/).

    a. To use service auto scaling, select **Service auto scaling**.

    b. For **Minimum number of tasks**, enter the lower limit of the number of tasks for service auto scaling to use. The desired count will not go below this count.

    c. For **Maximum number of tasks**, enter the upper limit of the number of tasks for service auto scaling to use. The desired count will not go above this count.

    d. Choose the policy type. Under **Scaling policy type**, choose one of the following options.

To use this policy type	Do this
Target tracking	a. For **Scaling policy type**, choose **Target tracking**. b. For **Policy name**, enter the name of the policy. c. For **ECS service metric**, select one of the following metrics. • **ECSServiceAverageCPUUtilization** – Average CPU utilization of the service. • **ECSServiceAverageMemoryUtilization** – Average memory

To use this policy type	Do this	
	utilization of the service. • **ALBRequestCountPerTarget** – Number of requests completed per target in an Application Load Balancer target group. d. For **Target value**, enter the value the service maintains for the selected metric. e. For **Scale-out cooldown period**, enter the amount of time, in seconds, after a scale-out activity (add tasks) that must pass before another scale-out activity can start. f. For **Scale-in cooldown period**, enter the amount of time, in seconds, after a scale-in activity (remove tasks) that must pass before another scale-in activity can start. g. To prevent the policy from performing a scale-in activity, select **Turn off scale-in**. h. • (Optional) Select **Turn off scale-in** if you want	

To use this policy type	Do this	
	your scaling policy to scale out for increased traffic but don't need it to scale in when traffic decreases.	

To use this policy type	Do this	
Step scaling	a. For **Scaling policy type**, choose **Step scaling**. b. For **Policy name**, enter the policy name. c. For **Alarm name**, enter a unique name for the alarm. d. For **Amazon ECS service metric**, choose the metric to use for the alarm. e. For **Statistic**, choose the alarm statistic. f. For **Period**, choose the period for the alarm. g. For **Alarm condition**, choose how to compare the selected metric to the defined threshold. h. For **Threshold to compare metrics** and **Evaluation period to initiate alarm**, enter the threshold used for the alarm and how long to evaluate the threshold. i. Under **Scaling actions**, do the following: • For **Action**, select whether to add, remove, or set a specific desired count for your service.	

To use this policy type	Do this	
	• If you chose to add or remove tasks, for **Value**, enter the number of tasks (or percent of existing tasks) to add or remove when the scaling action is initiated. If you chose to set the desired count, enter the number of tasks. For **Type**, select whether the **Value** is an integer or a percent value of the existing desired count. • For **Lower bound** and **Upper bound**, enter the lower boundary and upper boundary of your step scaling adjustmen t. By default, the lower bound for an add policy is the alarm threshold and the upper bound is positive (+) infinity. By default, the upper bound for a remove policy is the alarm threshold and	

To use this policy type	Do this	
	the lower bound is negative (-) infinity. • (Optional) Add additional scaling options. Choose **Add new scaling action**, and then repeat the **Scaling actions** steps. • For **Cooldown period**, enter the amount of time, in seconds, to wait for a previous scaling activity to take effect. For an add policy, this is the time after a scale-out activity that the scaling policy blocks scale-in activities and limits how many tasks can be scale out at a time. For a remove policy, this is the time after a scale-in activity that must pass before another scale-in activity can start.	

14. (Optional) To use a task placement strategy other than the default, expand **Task Placement**, and then choose from the following options.

    For more information, see [How Amazon ECS places tasks on container instances](#).

- **AZ Balanced Spread –** Distribute tasks across Availability Zones and across container instances in the Availability Zone.
- **AZ Balanced BinPack –** Distribute tasks across Availability Zones and across container instances with the least available memory.
- **BinPack –** Distribute tasks based on the least available amount of CPU or memory.
- **One Task Per Host –** Place, at most, one task from the service on each container instance.
- **Custom –** Define your own task placement strategy.

If you chose **Custom**, define the algorithm for placing tasks and the rules that are considered during task placement.

- Under **Strategy**, for **Type** and **Field**, choose the algorithm and the entity to use for the algorithm.

  You can enter a maximum of 5 strategies.
- Under **Constraint**, for **Type** and **Expression**, choose the rule and attribute for the constraint.

  For example, to set the constraint to place tasks on T2 instances, for the **Expression**, enter **attribute:ecs.instance-type =~ t2.***.

  You can enter a maximum of 10 constraints.

15. If your task uses a data volume that's compatible with configuration at deployment, you can configure the volume by expanding **Volume**.

    The volume name and volume type are configured when you create a task definition revision and can't be changed when creating a service. To update the volume name and type, you must create a new task definition revision and create a service by using the new revision.

To configure this volume type	Do this	
Amazon EBS	a. For **EBS volume type**, choose the type of EBS volume that you want to attach to your task.	

To configure this volume type	Do this	
	b. For **Size (GiB)**, enter a valid value for the volume size in gibibytes (GiB). You can specify a minimum of 1 GiB and a maximum of 16,384 GiB volume size. This value is required unless you provide a snapshot ID. c. For **IOPS**, enter the maximum number of input/output operation s (IOPS) that the volume should provide. This value is configurable only for `io1`,`io2`, and `gp3` volume types. d. For **Throughput (MiB/s)**, enter the throughpu t that the volume should provide, in mebibytes per second (MiBps, or MiB/s). This value is configurable only for the `gp3` volume type. e. For **Snapshot ID**, choose an existing Amazon EBS volume snapshot or enter the ARN of a snapshot if you want to create a volume from a snapshot. You can also create a new, empty volume by not	

To configure this volume type	Do this	
	choosing or entering a snapshot ID. f. If you specify a **Snapshot ID**, you can specify a **Volume initialization rate (MiB/s)**. Enter a value between 100 and 300, in MiB/s, that will determine how fast data is loaded from the snapshot specified using **Snapshot ID** for volume creation. g. For **File system type**, choose the type of file system that will be used for data storage and retrieval on the volume. You can choose either the operating system default or a specific file system type. The default for Linux is XFS. For volumes created from a snapshot, you must specify the same filesystem type that the volume was using when the snapshot was created. If there is a filesystem type mismatch, the task will fail to start.	

To configure this volume type	Do this	
	h. For **Infrastructure role**, choose an IAM role with the necessary permissions that allow Amazon ECS to manage Amazon EBS volumes for tasks. You can attach the `AmazonECSInfrastructureRolePolicyForVolumes` managed policy to the role, or you can use the policy as a guide to create and attach an your own policy with permissions that meet your specific needs. For more information about the necessary permissions, see Amazon ECS infrastructure IAM role. i. For **Encryption**, choose **Default** if you want to use the Amazon EBS encryption by default settings. If your account has Encryption by default configured, the volume will be encrypted with the AWS Key Management Service (AWS KMS) key that's specified in the setting. If you choose	

To configure this volume type	Do this	
	Default and Amazon EBS default encryption isn't turned on, the volume will be unencrypted. If you choose **Custom**, you can specify an AWS KMS key of your choice for volume encryption. If you choose **None**, the volume will be unencrypted unless you have encryption by default configured, or if you create a volume from an encrypted snapshot. j. If you've chosen **Custom** for **Encryption**, you must specify the AWS KMS key that you want to use. For **KMS key**, choose an AWS KMS key or enter a key ARN. If you choose to encrypt your volume by using a symmetric customer managed key, make sure that you have the right permissions defined in your AWS KMS key policy. For more information, see Data encryption for Amazon EBS volumes.	

To configure this volume type	Do this	
	k. (Optional) Under **Tags**, you can add tags to your Amazon EBS volume by either propagating tags from the task definition or service, or by providing your own tags. If you want to propagate tags from the task definition, choose **Task definition** for **Propagate tags from**. If you want to propagate tags from the service, choose **Service** for **Propagate tags from**. If you choose **Do not propagate**, or if you don't choose a value, the tags aren't propagated. If you want to provide your own tags, choose **Add tag** and then provide the key and value for each tag you add. For more information about tagging Amazon EBS volumes, see [Tagging Amazon EBS volumes].	

16. To use ECS Exec to debug the service, under **Troubleshooting configuration**, select **Turn on ECS Exec**.

17. (Optional) To help identify your service and tasks, expand the **Tags** section, and then configure your tags.

    To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the task definition tags, select **Turn on Amazon ECS managed tags**, and then for **Propagate tags from**, choose **Task definitions**.

    To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the service tags, select **Turn on Amazon ECS managed tags**, and then for **Propagate tags from**, choose **Service**.

    Add or remove a tag.

    - [Add a tag] Choose **Add tag**, and then do the following:
      - For **Key**, enter the key name.
      - For **Value**, enter the key value.
    - [Remove a tag] Next to the tag, choose **Remove tag**.
18. Choose **Create**.

Next steps

The following are additional actions after you create a service.

- Configure predicte auto scaling, which looks at past load data from traffic flows. For more information, see [Use historical patterns to scale Amazon ECS services with predictive scaling]().
- Track your deployment and view your service history for services that Amazon ECS circuit breaker. For more information, see [View service history using Amazon ECS service deployments]().

Amazon ECS blue/green deployments

A blue/green deployment is a release methodology that reduces downtime and risk by running two identical production environments called blue and green. With Amazon ECS blue/green deployments, you can validate new service revisions before directing production traffic to them. This approach provides a safer way to deploy changes with the ability to quickly roll back if needed.

Benefits

The following are benefits of using blue/green deployments:

- Reduces risk through testing with production traffic before switching production. You can validate the new deployment with test traffic before directing production traffic to it.
- Zero downtime deployments. The production environment remains available throughout the deployment process, ensuring continuous service availability.
- Easy rollback if issues are detected. If problems arise with the green deployment, you can quickly revert to the blue deployment without extended service disruption.
- Controlled testing environment. The green environment provides an isolated space to test new features with real traffic patterns before full deployment.
- Predictable deployment process. The structured approach with defined lifecycle stages makes deployments more consistent and reliable.
- Automated validation through lifecycle hooks. You can implement automated tests at various stages of the deployment to verify functionality.

Terminology

The following are Amazon ECS blue/green deployment terms:

- Bake time - The duration when both blue and green service revisions are running simultaneously after the production traffic has shifted.
- Blue deployment - The current production service revision that you want to replace.
- Green deployment - The new service revision that you want to deploy.
- Lifecycle stages - A series of events in the deployment operation, such as "after production traffic shift".
- Lifecycle hook - A Lambda function that verifies the deployment at a specific lifecycle stage.
- Listener - A Elastic Load Balancing resource that checks for connection requests using the protocol and port that you configure. The rules that you define for a listener determine how Amazon ECS routes requests to its registered targets.
- Rule - An Elastic Load Balancing resource associated with a listener. A rule defines how requests are routed and consists of an action, condition, and priority.
- Target group - An Elastic Load Balancing resource used to route requests to one or more registered targets (for example, EC2 instances). When you create a listener, you specify a target group for its default action. Traffic is forwarded to the target group specified in the listener rule.

- Traffic shift - The process Amazon ECS uses to shift traffic from the blue deployment to the green deployment. For Amazon ECS blue/green deployments, all traffic is shifted from the blue service to the green service at once.

Considerations

Consider the following when choosing a deployment type:

- Resource usage: Blue/green deployments temporarily run both the blue and green service revisions simultaneously, which may double your resource usage during deployments.
- Deployment monitoring: Blue/green deployments provide more detailed deployment status information, allowing you to monitor each stage of the deployment process.
- Rollback: Blue/green deployments make it easier to roll back to the previous version if issues are detected, as the blue revision is kept running until the bake time expires.

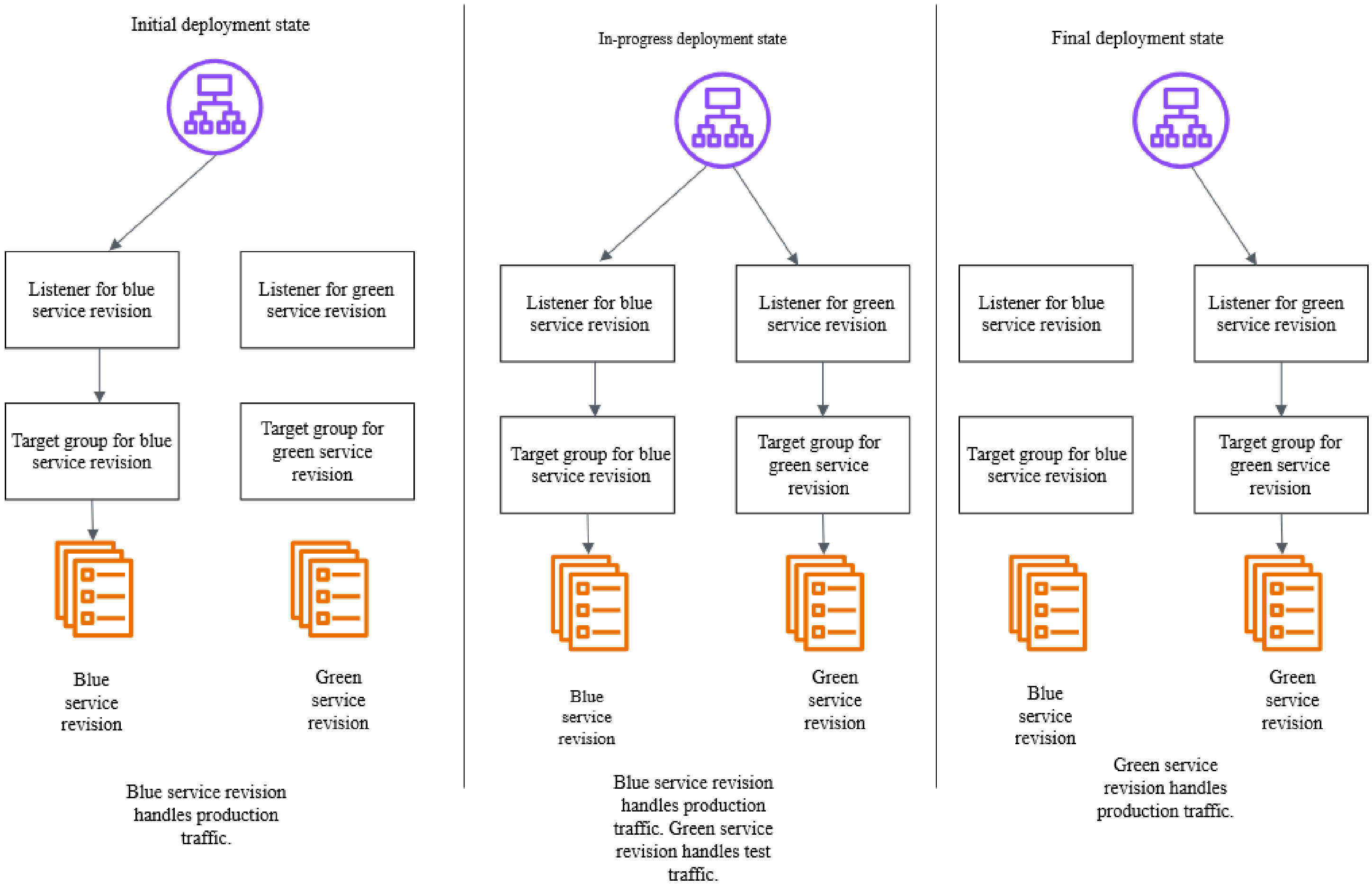
Amazon ECS blue/green service deployments workflow

The Amazon ECS blue/green deployment process follows a structured approach with six distinct phases that ensure safe and reliable application updates. Each phase serves a specific purpose in validating and transitioning your application from the current version (blue) to the new version (green).

1. **Preparation Phase**: Create the green environment alongside the existing blue environment. This includes provisioning new service revisions, and preparing target groups.
2. **Deployment Phase**: Deploy the new service revision to the green environment. Amazon ECS launches new tasks using the updated service revision while the blue environment continues serving production traffic.
3. **Testing Phase**: Validate the green environment using test traffic routing. The Application Load Balancer directs test requests to the green environment while production traffic remains on blue.
4. **Traffic Shifting Phase**: Shift production traffic from blue to green based on your configured deployment strategy. This phase includes monitoring and validation checkpoints.
5. **Monitoring Phase**: Monitor application health, performance metrics, and alarm states during the bake time period. A rollback operation is initiated when issues are detected.
6. **Completion Phase**: Finalize the deployment by terminating the blue environment or maintaining it for potential rollback scenarios, depending on your configuration.

Workflow

The following diagram illustrates the comprehensive blue/green deployment workflow, showing the interaction between Amazon ECS, and the Application Load Balancer:



The enhanced deployment workflow includes the following detailed steps:

1. **Initial State**: The blue service (current production) handles 100% of production traffic. The Application Load Balancer has a single listener with rules that route all requests to the blue target group containing healthy blue tasks.
2. **Green Environment Provisioning**: Amazon ECS creates new tasks using the updated task definition. These tasks are registered with a new green target group but receive no traffic initially.
3. **Health Check Validation**: The Application Load Balancer performs health checks on green tasks. Only when green tasks pass health checks does the deployment proceed to the next phase.
4. **Test Traffic Routing**: If configured, the Application Load Balancer's listener rules route specific traffic patterns (such as requests with test headers) to the green environment for validation

while production traffic remains on blue. This is controlled by the same listener that handles production traffic, using different rules based on request attributes.

5. **Production Traffic Shift**: Based on the deployment configuration, traffic shifts from blue to green. In ECS blue/green deployments, this is an immediate (all-at-once) shift where 100% of the traffic is moved from the blue to the green environment. The Application Load Balancer uses a single listener with listener rules that control traffic distribution between the blue and green target groups based on weights.
6. **Monitoring and Validation**: Throughout the traffic shift, Amazon ECS monitors CloudWatch metrics, alarm states, and deployment health. Automatic rollback triggers activate if issues are detected.
7. **Bake Time Period**: The duration when both blue and green service revisions are running simultaneously after the production traffic has shifted.
8. **Blue Environment Termination**: After successful traffic shift and validation, the blue environment is terminated to free up cluster resources, or maintained for rapid rollback capability.
9. **Final State**: The green environment becomes the new production environment, handling 100% of traffic. The deployment is marked as successful.

Deployment lifecycle stages

The blue/green deployment process progresses through distinct lifecycle stages (a series of events in the deployment operation, such as "after production traffic shift"), each with specific responsibilities and validation checkpoints. Understanding these stages helps you monitor deployment progress and troubleshoot issues effectively.

Each lifecycle stage can last up to 24 hours. We recommend that the value remains below the 24-hour mark. This is because asynchronous processes need time to trigger the hooks. The system times out, fails the deployment, and then initiates a rollback after a stage reaches 24 hours. AWS CloudFormation deployments have additional timeout restrictions. While the 24-hour stage limit remains in effect, AWS CloudFormation enforces a 36-hour limit on the entire deployment. AWS CloudFormation fails the deployment, and then initiates a rollback if the process doesn't complete within 36 hours.

Lifecycle stages	Description	Use this stage for lifecycle hook?
RECONCILE_SERVICE	This stage only happens when you start a new service deployment with more than 1 service revision in an ACTIVE state.	Yes
PRE_SCALE_UP	The green service revision has not started. The blue service revision is handling 100% of the production traffic. There is no test traffic.	Yes
SCALE_UP	The time when the green service revision scales up to 100% and launches new tasks. The green service revision is not serving any traffic at this point.	No
POST_SCALE_UP	The green service revision has started. The blue service revision is handling 100% of the production traffic. There is no test traffic.	Yes
TEST_TRAFFIC_SHIFT	The blue and green service revisions are running. The blue service revision handles 100% of the production traffic. The green service revision is migrating from 0% to 100% of test traffic.	Yes
POST_TEST_TRAFFIC_SHIFT	The test traffic shift is complete. The green service revision handles 100% of the test traffic.	Yes
PRODUCTION_TRAFFIC _SHIFT	Production traffic is shifting to the green service revision. The green service revision is migrating from 0% to 100% of production traffic.	Yes

Lifecycle stages	Description	Use this stage for lifecycle hook?
POST_PRODUCTION_TR AFFIC_SHIFT	The production traffic shift is complete.	Yes
BAKE_TIME	The duration when both blue and green service revisions are running simultane ously.	No
CLEAN_UP	The blue service revision has completel y scaled down to 0 running tasks. The green service revision is now the production service revision after this stage.	No

Each lifecycle stage includes built-in validation checkpoints that must pass before proceeding to the next stage. If any validation fails, the deployment can be automatically rolled back to maintain service availability and reliability.

When you use a Lambda function, the function must complete the work, or return IN_PROGRESS within 15 minutes. You can use the `callBackDelaySeconds` to delay the call to Lambda. For more information, see app.py function in the sample-amazon-ecs-blue-green-deployment-patterns on GitHub.

Required resources for Amazon ECS blue/green deployments

To use a blue/green deployment with managed traffic shifting, your service must use one of the following features:

- Elastic Load Balancing
- Service Connect

Services that don't use Service Discovery, Service Connect, VPC Lattice or Elastic Load Balancing can also use blue/green deployments, but don't get any of the managed traffic shifting benefits.

The following list provides a high-level overview of what you need to configure for Amazon ECS blue/green deployments:

- Your service uses an Application Load Balancer, Network Load Balancer, or Service Connect. Configure the appropriate resources.
  - Application Load Balancer - For more information, see Application Load Balancer resources for blue/green deployments.
  - Network Load Balancer - For more information, see Network Load Balancer resources for Amazon ECS blue/green deployments.
  - Service Connect - For more information, see Service Connect resources for Amazon ECS blue/green deployments.
- Set the service deployment controller to `ECS`.
- Configure the deployment strategy as `blue/green` in your service definition.
- Optionally, configure additional parameters such as:
  - Bake time for the new deployment
  - CloudWatch alarms for automatic rollback
  - Deployment lifecycle hooks for testing (these are Lambda functions that run at specified deployment stages)

Best practices

Follow these best practices for successful Amazon ECS blue/green deployments:

- Configure appropriate health checks that accurately reflect your application's health.
- Set a bake time that allows sufficient testing of the green deployment.
- Implement CloudWatch alarms to automatically detect issues and trigger rollbacks.
- Use lifecycle hooks to perform automated testing at each deployment stage.
- Ensure your application can handle both blue and green environments running simultaneously.
- Plan for sufficient cluster capacity to handle both environments during deployment.
- Test your rollback procedures before implementing them in production.

Application Load Balancer resources for blue/green deployments

To use Application Load Balancers with Amazon ECS blue/green deployments, you need to configure specific resources that allow traffic routing between the blue and green service revisions.

Target groups

For blue/green deployments with Elastic Load Balancing, you need to create two target groups:

- A primary target group for the blue service revision (current production traffic)
- An alternate target group for the green service revision (new version)

Both target groups should be configured with the following settings:

- Target type: `IP` (for Fargate or EC2 with `awsvpc` network mode)
- Protocol: `HTTP` (or the protocol your application uses)
- Port: The port your application listens on (typically `80` for HTTP)
- VPC: The same VPC as your Amazon ECS tasks
- Health check settings: Configured to properly check your application's health

During a blue/green deployment, Amazon ECS automatically registers tasks with the appropriate target group based on the deployment stage.

Example Creating target groups for an Application Load Balancer

The following CLI commands create two target groups for use with an Application Load Balancer in a blue/green deployment:

```
aws elbv2 create-target-group \
    --name blue-target-group \
    --protocol HTTP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-path / \
    --health-check-protocol HTTP \
    --health-check-interval-seconds 30 \
    --health-check-timeout-seconds 5 \
    --healthy-threshold-count 2 \
    --unhealthy-threshold-count 2

aws elbv2 create-target-group \
    --name green-target-group \
    --protocol HTTP \
    --port 80 \
```

```
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-path / \
    --health-check-protocol HTTP \
    --health-check-interval-seconds 30 \
    --health-check-timeout-seconds 5 \
    --healthy-threshold-count 2 \
    --unhealthy-threshold-count 2
```

Application Load Balancer

You need to create an Application Load Balancer with the following configuration:

- Scheme: Internet-facing or internal, depending on your requirements
- IP address type: IPv4
- VPC: The same VPC as your Amazon ECS tasks
- Subnets: At least two subnets in different Availability Zones
- Security groups: A security group that allows traffic on the listener ports

The security group attached to the Application Load Balancer must have an outbound rule that allows traffic to the security group attached to your Amazon ECS tasks.

Example Creating an Application Load Balancer

The following CLI command creates anApplication Load Balancer for use in a blue/green deployment:

```
aws elbv2 create-load-balancer \
    --name my-application-load-balancer \
    --type application \
    --security-groups sg-abcd1234 \
    --subnets subnet-12345678 subnet-87654321
```

Listeners and rules

For blue/green deployments, you need to configure listeners on your Application Load Balancer:

- Production listener: Handles production traffic (typically on port 80 or 443)
  - Initially forwards traffic to the primary target group (blue service revision)
  - After deployment, forwards traffic to the alternate target group (green service revision)

- Test listener (optional): Handles test traffic to validate the green service revision before shifting production traffic
  - Can be configured on a different port (for example, 8080 or 8443)
  - Forwards traffic to the alternate target group (green service revision) during testing

During a blue/green deployment, Amazon ECS automatically updates the listener rules to route traffic to the appropriate target group based on the deployment stage.

Example Creating a production listener

The following CLI command creates a production listener on port 80 that forwards traffic to the primary (blue) target group:

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
app/my-application-load-balancer/abcdef123456 \
    --protocol HTTP \
    --port 80 \
    --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
blue-target-group/abcdef123456
```

Example Creating a test listener

The following CLI command creates a test listener on port 8080 that forwards traffic to the alternate (green) target group:

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
app/my-application-load-balancer/abcdef123456 \
    --protocol HTTP \
    --port 8080 \
    --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

Example Creating a listener rule for path-based routing

The following CLI command creates a rule that forwards traffic for a specific path to the green target group for testing:

```
aws elbv2 create-rule \
    --listener-arn arn:aws:elasticloadbalancing:region:123456789012:listener/app/my-
application-load-balancer/abcdef123456/ghijkl789012 \
    --priority 10 \
    --conditions Field=path-pattern,Values='/test/*' \
    --actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

Example Creating a listener rule for header-based routing

The following CLI command creates a rule that forwards traffic with a specific header to the green target group for testing:

```
aws elbv2 create-rule \
    --listener-arn arn:aws:elasticloadbalancing:region:123456789012:listener/app/my-
application-load-balancer/abcdef123456/ghijkl789012 \
    --priority 20 \
    --conditions Field=http-header,HttpHeaderConfig='{Name=X-
Environment,Values=[test]}' \
    --actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
green-target-group/ghijkl789012
```

Service configuration

You must have permissions to allow Amazon ECS to manage load balancer resources in your clusters on your behalf. For more information, see [Amazon ECS infrastructure IAM role for load balancers](#).

When creating or updating an Amazon ECS service for blue/green deployments with Elastic Load Balancing, you need to specify the following configuration.

Replace the *`user-input`* with your values.

The key components in this configuration are:

- `targetGroupArn`: The ARN of the primary target group (blue service revision).
- `alternateTargetGroupArn`: The ARN of the alternate target group (green service revision).
- `productionListenerRule`: The ARN of the listener rule for production traffic.

- `roleArn`: The ARN of the role that allows Amazon ECS to manage Elastic Load Balancing resources.
- `strategy`: Set to `BLUE_GREEN` to enable blue/green deployments.
- `bakeTimeInMinutes`: The duration when both blue and green service revisions are running simultaneously after the production traffic has shifted.
- `TestListenerRule`: The ARN of the listener rule for test traffic. This is an optional parameter.

```
{
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/primary-target-group/
abcdef123456",
            "containerName": "container-name",
            "containerPort": 80,
            "advancedConfiguration": {
                "alternateTargetGroupArn":
 "arn:aws:elasticloadbalancing:region:account-id:targetgroup/alternate-target-group/
ghijkl789012",
                "productionListenerRule": "arn:aws:elasticloadbalancing:region:account-
id:listener-rule/app/load-balancer-name/abcdef123456/listener/ghijkl789012/rule/
mnopqr345678",
                "roleArn": "arn:aws:iam::123456789012:role/ecs-elb-role"
            }
        }
    ],
    "deploymentConfiguration": {
        "strategy": "BLUE_GREEN",
        "maximumPercent": 200,
        "minimumHealthyPercent": 100,
        "bakeTimeInMinutes": 5
    }
}
```

Traffic flow during deployment

During a blue/green deployment with Elastic Load Balancing, traffic flows through the system as follows:

1. *Initial state*: All production traffic is routed to the primary target group (blue service revision).

2. *Green service revision deployment*: Amazon ECS deploys the new tasks and registers them with the alternate target group.
3. *Test traffic*: If a test listener is configured, test traffic is routed to the alternate target group to validate the green service revision.
4. *Production traffic shift*: Amazon ECS updates the production listener rule to route traffic to the alternate target group (green service revision).
5. *Bake time*: The duration when both blue and green service revisions are running simultaneously after the production traffic has shifted.
6. *Completion*: After a successful deployment, the blue service revision is terminated.

If issues are detected during the deployment, Amazon ECS can automatically roll back by routing traffic back to the primary target group (blue service revision).

Network Load Balancer resources for Amazon ECS blue/green deployments

To use a Network Load Balancer with Amazon ECS blue/green deployments, you need to configure specific resources that enable traffic routing between the blue and green service revisions. This section explains the required components and their configuration.

When your configuration includes a Network Load Balancer, Amazon ECS adds a 10 minute delay to the following lifecycle stages:

- PRE_SCALE_UP
- TEST_TRAFFIC_SHIFT
- PRODUCTION_TRAFFIC_SHIFT

This delay accounts for Network Load Balancer timing issues that can cause a mismatch between the configured traffic weights and the actual traffic routing in the data plane.

Target groups

For blue/green deployments with a Network Load Balancer, you need to create two target groups:

- A primary target group for the blue service revision (current production traffic)
- An alternate target group for the green service revision (new service revision)

Both target groups should be configured with the following settings:

- Target type: `ip` (for Fargate or EC2 with `awsvpc` network mode)
- Protocol: TCP (or the protocol your application uses)
- Port: The port your application listens on (typically `80` for HTTP)
- VPC: The same VPC as your Amazon ECS tasks
- Health check settings: Configured to properly check your application's health

  For TCP health checks, the Network Load Balancer establishes a TCP connection with the target. If the connection is successful, the target is considered healthy.

  For HTTP/HTTPS health checks, the Network Load Balancer sends an HTTP/HTTPS request to the target and verifies the response.

During a blue/green deployment, Amazon ECS automatically registers tasks with the appropriate target group based on the deployment stage.

Example Creating target groups for a Network Load Balancer

The following AWS CLI commands create two target groups for use with a Network Load Balancer in a blue/green deployment:

```
aws elbv2 create-target-group \
    --name blue-target-group \
    --protocol TCP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-protocol TCP

aws elbv2 create-target-group \
    --name green-target-group \
    --protocol TCP \
    --port 80 \
    --vpc-id vpc-abcd1234 \
    --target-type ip \
    --health-check-protocol TCP
```

Network Load Balancer

You need to create a Network Load Balancer with the following configuration:

- Scheme: Internet-facing or internal, depending on your requirements
- IP address type: IPv4
- VPC: The same VPC as your Amazon ECS tasks
- Subnets: At least two subnets in different Availability Zones

Unlike Application Load Balancers, Network Load Balancers operate at the transport layer (Layer 4) and do not use security groups. Instead, you need to ensure that the security groups associated with your Amazon ECS tasks allow traffic from the Network Load Balancer on the listener ports.

Example Creating a Network Load Balancer

The following AWS CLI command creates a Network Load Balancer for use in a blue/green deployment:

```
aws elbv2 create-load-balancer \
    --name my-network-load-balancer \
    --type network \
    --subnets subnet-12345678 subnet-87654321
```

Considerations for using NLB with blue/green deployments

When using a Network Load Balancer for blue/green deployments, consider the following:

- **Layer 4 operation**: Network Load Balancers operate at the transport layer (Layer 4) and do not inspect application layer (Layer 7) content. This means you cannot use HTTP headers or paths for routing decisions.
- **Health checks**: Network Load Balancer health checks are limited to TCP, HTTP, or HTTPS protocols. For TCP health checks, the Network Load Balancer only verifies that the connection can be established.
- **Connection preservation**: Network Load Balancers preserve the source IP address of the client, which can be useful for security and logging purposes.
- **Static IP addresses**: Network Load Balancers provide static IP addresses for each subnet, which can be useful for whitelisting or when clients need to connect to a fixed IP address.
- **Test traffic**: Since Network Load Balancers do not support content-based routing, test traffic must be sent to a different port than production traffic.

Listeners and rules

For blue/green deployments with a Network Load Balancer, you need to configure listeners:

- Production listener: Handles production traffic (typically on port 80 or 443)
  - Initially forwards traffic to the primary target group (blue service revision)
  - After deployment, forwards traffic to the alternate target group (green service revision)
- Test listener (required): Handles test traffic to validate the green service revision before shifting production traffic
  - Can be configured on a different port (e.g., 8080 or 8443)
  - Forwards traffic to the alternate target group (green service revision) during testing

Unlike Application Load Balancers, Network Load Balancers do not support content-based routing rules. Instead, traffic is routed based on the listener port and protocol.

The following AWS CLI commands create production and test listeners for a Network Load Balancer:

Replace the *user-input* with your values.

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
net/my-network-lb/1234567890123456 \
    --protocol TCP \
    --port 80 \
    --default-actions Type=forward,
 TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/blue-
target-group/1234567890123456

aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:region:123456789012:loadbalancer/
net/my-network-lb/1234567890123456 \
    --protocol TCP \
    --port 8080 \
    --default-actions Type=forward,
 TargetGroupArn=arn:aws:elasticloadbalancing:region:123456789012:targetgroup/green-
target-group/1234567890123456
```

Service configuration

You must have permissions to allow Amazon ECS to manage load balancer resources in your clusters on your behalf. For more information, see Amazon ECS infrastructure IAM role for load balancers.

When creating or updating an Amazon ECS service for blue/green deployments with a Network Load Balancer, you need to specify the following configuration:

Replace the *user-input* with your values.

The key components in this configuration are:

- `targetGroupArn`: The ARN of the primary target group (blue service revision)
- `alternateTargetGroupArn`: The ARN of the alternate target group (green service revision)
- `productionListenerRule`: The ARN of the listener for production traffic
- `testListenerRule`: (Optional) The ARN of the listener for test traffic
- `roleArn`: The ARN of the role that allows Amazon ECS to manage Network Load Balancer resources
- `strategy`: Set to `BLUE_GREEN` to enable blue/green deployments
- `bakeTimeInMinutes`: The duration to wait after the green service revision is deployed before shifting production traffic

```
{
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/blue-target-
group/1234567890123456",
            "containerName": "container-name",
            "containerPort": 80,
            "advancedConfiguration": {
                "alternateTargetGroupArn":
 "arn:aws:elasticloadbalancing:region:123456789012:targetgroup/green-target-
group/1234567890123456",
                "productionListenerRule":
 "arn:aws:elasticloadbalancing:region:123456789012:listener/net/my-network-
lb/1234567890123456/1234567890123456",
```

```
                "testListenerRule":
 "arn:aws:elasticloadbalancing:region:123456789012:listener/net/my-network-
lb/1234567890123456/2345678901234567",
                "roleArn": "arn:aws:iam::123456789012:role/ecs-nlb-role"
            }
        }
    ],
    "deploymentConfiguration": {
        "strategy": "BLUE_GREEN",
        "maximumPercent": 200,
        "minimumHealthyPercent": 100,
        "bakeTimeInMinutes": 5
    }
}
```

Traffic flow during deployment

During a blue/green deployment with a Network Load Balancer, traffic flows through the system as follows:

1. *Initial state*: All production traffic is routed to the primary target group (blue service revision).
2. *Green service revision deployment*: Amazon ECS deploys the new tasks and registers them with the alternate target group.
3. *Test traffic*: If a test listener is configured, test traffic is routed to the alternate target group to validate the green service revision.
4. *Production traffic shift*: Amazon ECS updates the production listener to route traffic to the alternate target group (green service revision).
5. *Bake time*: The duration when both blue and green service revisions are running simultaneously after the production traffic has shifted.
6. *Completion*: After a successful deployment, the blue service revision is terminated.

If issues are detected during the deployment, Amazon ECS can automatically roll back by routing traffic back to the primary target group (blue service revision).

Service Connect resources for Amazon ECS blue/green deployments

When using Service Connect with blue/green deployments, you need to configure specific components to enable proper traffic routing between the blue and green service revisions. This section explains the required components and their configuration.

Architecture overview

Service Connect builds both service discovery and service mesh capabilities through a managed sidecar proxy that's automatically injected into your Amazon ECS tasks. These proxies handle routing decisions, retries, and metrics collection, while AWS Cloud Map provides the service registry backend. When you deploy a service with Service Connect enabled, the service registers itself in AWS Cloud Map, and client services discover it through the namespace.

In a standard Service Connect implementation, client services connect to logical service names, and the sidecar proxy handles routing to the actual service instances. With blue/green deployments, this model is extended to include test traffic routing through the `testTrafficRules` configuration.

During a blue/green deployment, the following key components work together:

- **Service Connect Proxy**: All traffic between services passes through the Service Connect proxy, which makes routing decisions based on the configuration.
- **AWS Cloud Map Registration**: Both blue and green deployments register with AWS Cloud Map, but the green deployment initially registers as a "test" endpoint.
- **Test Traffic Routing**: The `testTrafficRules` in the Service Connect configuration determine how to identify and route test traffic to the green deployment. This is accomplished through **header-based routing**, where specific HTTP headers in the requests direct traffic to the test revision. By default, Service Connect recognizes the `x-amzn-ecs-blue-green-test` header for HTTP-based protocols when no custom rules are specified.
- **Client Configuration**: All clients in the namespace automatically receive both production and test routes, but only requests matching test rules will go to the green deployment.

What makes this approach powerful is that it handles the complexity of service discovery during transitions. As traffic shifts from the blue to green deployment, all connectivity and discovery mechanisms update automatically. There's no need to update DNS records, reconfigure load balancers, or deploy service discovery changes separately since the service mesh handles it all.

Traffic routing and testing

Service Connect provides advanced traffic routing capabilities for blue/green deployments, including header-based routing and client alias configuration for testing scenarios.

Test traffic header rules

During blue/green deployments, you can configure test traffic header rules to route specific requests to the green (new) service revision for testing purposes. This allows you to validate the new version with controlled traffic before completing the deployment.

Service Connect uses **header-based routing** to identify test traffic. By default, Service Connect recognizes the `x-amzn-ecs-blue-green-test` header for HTTP-based protocols when no custom rules are specified. When this header is present in a request, the Service Connect proxy automatically routes the request to the green deployment for testing.

Test traffic header rules enable you to:

- Route requests with specific headers to the green service revision
- Test new functionality with a subset of traffic
- Validate service behavior before full traffic cutover
- Implement canary testing strategies
- Perform integration testing in a production-like environment

The header-based routing mechanism works seamlessly with your existing application architecture. Client services don't need to be aware of the blue/green deployment process - they simply include the appropriate headers when sending test requests, and the Service Connect proxy handles the routing logic automatically.

For more information about configuring test traffic header rules, see ServiceConnectTestTrafficHeaderRules in the *Amazon Elastic Container Service API Reference*.

Header matching rules

Header matching rules define the criteria for routing test traffic during blue/green deployments. You can configure multiple matching conditions to precisely control which requests are routed to the green service revision.

Header matching supports:

- Exact header value matching
- Header presence checking
- Pattern-based matching
- Multiple header combinations

Example use cases include routing requests with specific user agent strings, API versions, or feature flags to the green service for testing.

For more information about header matching configuration, see ServiceConnectTestTrafficHeaderMatchRules in the *Amazon Elastic Container Service API Reference*.

Client aliases for blue/green deployments

Client aliases provide stable DNS endpoints for services during blue/green deployments. They enable seamless traffic routing between blue and green service revisions without requiring client applications to change their connection endpoints.

During a blue/green deployment, client aliases:

- Maintain consistent DNS names for client connections
- Enable automatic traffic switching between service revisions
- Support gradual traffic migration strategies
- Provide rollback capabilities by redirecting traffic to the blue revision

You can configure multiple client aliases for different ports or protocols, allowing complex service architectures to maintain connectivity during deployments.

For more information about client alias configuration, see ServiceConnectClientAlias in the *Amazon Elastic Container Service API Reference*.

Best practices for traffic routing

When implementing traffic routing for blue/green deployments with Service Connect, consider the following best practices:

- **Start with header-based testing**: Use test traffic header rules to validate the green service with controlled traffic before switching all traffic.
- **Configure health checks**: Ensure both blue and green services have appropriate health checks configured to prevent routing traffic to unhealthy instances.
- **Monitor service metrics**: Track key performance indicators for both service revisions during the deployment to identify issues early.
- **Plan rollback strategy**: Configure client aliases and routing rules to enable quick rollback to the blue service if issues are detected.

- **Test header matching logic**: Validate your header matching rules in a non-production environment before applying them to production deployments.

Service Connect blue/green deployment workflow

Understanding how Service Connect manages the blue/green deployment process helps you implement and troubleshoot your deployments effectively. The following workflow shows how the different components interact during each phase of the deployment.

Deployment phases

A Service Connect blue/green deployment progresses through several distinct phases:

1. **Initial State**: The blue service handles 100% of production traffic. All client services in the namespace connect to the blue service through the logical service name configured in Service Connect.
2. **Green Service Registration**: When the green deployment starts, it registers with AWS Cloud Map as a "test" endpoint. The Service Connect proxy in client services automatically receives both production and test route configurations.
3. **Test Traffic Routing**: Requests containing the test traffic headers (such as `x-amzn-ecs-blue-green-test`) are automatically routed to the green service by the Service Connect proxy. Production traffic continues to flow to the blue service.
4. **Traffic Shift Preparation**: After successful testing, the deployment process prepares for production traffic shift. Both blue and green services remain registered and healthy.
5. **Production Traffic Shift**: The Service Connect configuration updates to route production traffic to the green service. This happens automatically without requiring client service updates or DNS changes.
6. **Bake Time Period**: The duration when both blue and green service revisions are running simultaneously after the production traffic has shifted.
7. **Blue Service Deregistration**: After successful traffic shift and validation, the blue service is deregistered from AWS Cloud Map and terminated, completing the deployment.

Service Connect proxy behavior

The Service Connect proxy plays a crucial role in managing traffic during blue/green deployments. Understanding its behavior helps you design effective testing and deployment strategies.

Key proxy behaviors during blue/green deployments:

- **Automatic Route Discovery**: The proxy automatically discovers both production and test routes from AWS Cloud Map without requiring application restarts or configuration changes.
- **Header-Based Routing**: The proxy examines incoming request headers and routes traffic to the appropriate service revision based on the configured test traffic rules.
- **Health Check Integration**: The proxy only routes traffic to healthy service instances, automatically excluding unhealthy tasks from the routing pool.
- **Retry and Circuit Breaking**: The proxy provides built-in retry logic and circuit breaking capabilities, improving resilience during deployments.
- **Metrics Collection**: The proxy collects detailed metrics for both blue and green services, enabling comprehensive monitoring during deployments.

Service discovery updates

One of the key advantages of using Service Connect for blue/green deployments is the automatic handling of service discovery updates. Traditional blue/green deployments often require complex DNS updates or load balancer reconfiguration, but Service Connect manages these changes transparently.

During a deployment, Service Connect handles:

- **Namespace Updates**: The Service Connect namespace automatically includes both blue and green service endpoints, with appropriate routing rules.
- **Client Configuration**: All client services in the namespace automatically receive updated routing information without requiring restarts or redeployment.
- **Gradual Transition**: Service discovery updates happen gradually and safely, ensuring no disruption to ongoing requests.
- **Rollback Support**: If a rollback is needed, Service Connect can quickly revert service discovery configurations to route traffic back to the blue service.

Creating an Amazon ECS blue/green deployment

By using Amazon ECS blue/green deployments, you can make and test service changes before implementing them in a production environment.

Prerequisites

Perform the following operations before you start a blue/green deployment.

1. Configure the appropriate permissions.
   - For information about Elastic Load Balancing permissions, see Amazon ECS infrastructure IAM role for load balancers.
   - For information about Lambda permissions, see Permissions required for Lambda functions in Amazon ECS blue/green deployments
2. Amazon ECS blue/green deployments require that your service to use one of the following features: Configure the appropriate resources.
   - Application Load Balancer - For more information, see Application Load Balancer resources for blue/green deployments.
   - Network Load Balancer - For more information, see Network Load Balancer resources for Amazon ECS blue/green deployments.
   - Service Connect - For more information, see Service Connect resources for Amazon ECS blue/green deployments.
3. Create a rule to route traffic to your green service revision. For more information, see Listener rules in the *Network Load Balancer User Guide*.
4. Create a target group for your green service revision. When you use the `awsvpc` network mode for your tasks, the target type must be `ip`. For information about target groups, see Target groups in the *Network Load Balancer User Guide*.
5. Decide if you want to run Lambda functions for the lifecycle events.
   - Pre scale up
   - After scale up
   - Test traffic shift
   - After test traffic shift
   - Production traffic shift
   - After production traffic shift

   Create Lambda functions for each lifecycle event. For more information, see Create a Lambda function with the console in the *AWS Lambda Developer Guide*.

Procedure

You can use the console or the AWS CLI to create an Amazon ECS blue/green service.

Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. Determine the resource from where you launch the service.

To start a service from	Steps
Clusters	a. On the **Clusters** page, select the cluster to create the service in. The cluster details page displays. b. On the **Services** tab, choose **Create**.
Task definition	a. On the **Task definitions** page, select the task definition. b. From the **Deploy** menu, choose **Create service**.

The **Create service** page displays.

3. Under **Service details**, do the following:

   a. For **Task definition family**, choose the task definition to use. Then, for **Task definition revision**, enter the revision to use.

   b. For **Service name**, enter a name for your service.

4. To run the service in an existing cluster, for **Existing cluster**, choose the cluster. To run the service in a new cluster, choose **Create cluster**
5. Choose how your tasks are distributed across your cluster infrastructure. Under **Compute configuration**, choose your option.

Compute option	Steps	
Capacity provider strategy	a. Under **Compute options**, choose **Capacity provider strategy**. b. Choose a strategy: • To use the cluster's default capacity provider strategy, choose **Use cluster default**. • If your cluster doesn't have a default capacity provider strategy, or to use a custom strategy, choose **Use custom**, **Add capacity provider strategy**, and then define your custom capacity provider strategy by specifyin g a **Base**, **Capacity provider**, and **Weight**. **Note** To use a capacity provider in a strategy, the capacity provider	

Compute option	Steps
	must be associated with the cluster.
Launch type	a. In the **Compute options** section, select **Launch type**. b. For **Launch type**, choose a launch type. c. (Optional) When the Fargate launch type is specified, for **Platform version**, specify the platform version to use. If a platform version isn't specified , the `LATEST` platform version is used.

6. Under **Deployment configuration**, do the following:

   a. For **Service type**, choose **Replica**.

   b. For **Desired tasks**, enter the number of tasks to launch and maintain in the service.

   c. To have Amazon ECS monitor the distribution of tasks across Availability Zones, and redistribute them when there is an imbalance, under **Availability Zone service rebalancing**, select **Availability Zone service rebalancing**.

   d. For **Health check grace period**, enter the amount of time (in seconds) that the service scheduler ignores unhealthy Elastic Load Balancing, VPC Lattice, and container health checks after a task has first started. If you do not specify a health check grace period value, the default value of 0 is used.

7. a. For **Bake time**, enter the number of minutes that both the blue and green service revisions will run simultaneously before the blue revision is terminated. This allows time for verification and testing.

   b. (Optional) Run Lambda functions to run at specific stages of the deployment. Under **Deployment lifecycle hooks**, select the stages to run the lifecycle hooks.

To add a lifecycle hook:

a. Choose **Add**.
b. For **Lambda function**, enter the function name or ARN.
c. For **Role**, select the IAM role that has permission to invoke the Lambda function.
d. For **Lifecycle stages**, select the stages when the Lambda function should run.

8. To configure how Amazon ECS detects and handles deployment failures, expand **Deployment failure detection**, and then choose your options.

   a. To stop a deployment when the tasks cannot start, select **Use the Amazon ECS deployment circuit breaker**.

      To have the software automatically roll back the deployment to the last completed deployment state when the deployment circuit breaker sets the deployment to a failed state, select **Rollback on failures**.

   b. To stop a deployment based on application metrics, select **Use CloudWatch alarm(s)**. Then, from **CloudWatch alarm name**, choose the alarms. To create a new alarm, go to the CloudWatch console.

      To have the software automatically roll back the deployment to the last completed deployment state when a CloudWatch alarm sets the deployment to a failed state, select **Rollback on failures**.

9. (Optional) To interconnect your service using Service Connect, expand **Service Connect**, and then specify the following:

   a. Select **Turn on Service Connect**.
   b. Under **Service Connect configuration**, specify the client mode.

      - If your service runs a network client application that only needs to connect to other services in the namespace, choose **Client side only**.
      - If your service runs a network or web service application and needs to provide endpoints for this service, and connects to other services in the namespace, choose **Client and server**.

   c. To use a namespace that is not the default cluster namespace, for **Namespace**, choose the service namespace.

d. (Optional) Configure test traffic header rules for blue/green deployments. Under **Test traffic routing**, specify the following:

   i. Select **Enable test traffic header rules** to route specific requests to the green service revision during testing.

   ii. For **Header matching rules**, configure the criteria for routing test traffic:

      - **Header name**: Enter the name of the HTTP header to match (for example, `X-Test-Version` or `User-Agent`).
      - **Match type**: Choose the matching criteria:
        - **Exact match**: Route requests where the header value exactly matches the specified value
        - **Header present**: Route requests that contain the specified header, regardless of value
        - **Pattern match**: Route requests where the header value matches a specified pattern
      - **Header value** (if using exact match or pattern match): Enter the value or pattern to match against.

      You can add multiple header matching rules to create complex routing logic. Requests matching any of the configured rules will be routed to the green service revision for testing.

   iii. Choose **Add header rule** to configure additional header matching conditions.

> **Note**
>
> Test traffic header rules enable you to validate new functionality with controlled traffic before completing the full deployment. This allows you to test the green service revision with specific requests (such as those from internal testing tools or beta users) while maintaining normal traffic flow to the blue service revision.

e. (Optional) Specify a log configuration. Select **Use log collection**. The default option sends container logs to CloudWatch Logs. The other log driver options are configured using AWS FireLens. For more information, see [Send Amazon ECS logs to an AWS service or AWS Partner](#).

The following describes each container log destination in more detail.

- **Amazon CloudWatch** – Configure the task to send container logs to CloudWatch Logs. The default log driver options are provided, which create a CloudWatch log group on your behalf. To specify a different log group name, change the driver option values.
- **Amazon Data Firehose** – Configure the task to send container logs to Firehose. The default log driver options are provided, which send logs to a Firehose delivery stream. To specify a different delivery stream name, change the driver option values.
- **Amazon Kinesis Data Streams** – Configure the task to send container logs to Kinesis Data Streams. The default log driver options are provided, which send logs to an Kinesis Data Streams stream. To specify a different stream name, change the driver option values.
- **Amazon OpenSearch Service** – Configure the task to send container logs to an OpenSearch Service domain. The log driver options must be provided.
- **Amazon S3** – Configure the task to send container logs to an Amazon S3 bucket. The default log driver options are provided, but you must specify a valid Amazon S3 bucket name.

10. (Optional) Configure **Load balancing** for blue/green deployment.

Elastic Load Balancing type	Steps	
Application Load Balancer	a. For **Load balancer type**, choose **Application Load Balancer**. b. Choose **Create a new load balancer** to create a new Applicati on Load Balancer or **Use an existing load balancer** to select an	

Elastic Load Balancing type	Steps	
	existing Application Load Balancer. c. For **Container**, choose the container that hosts the service. d. For **Load balancer name**, enter a unique name. e. For **Listener**, enter a port and protocol for the Application Load Balancer to listen for connection requests on. By default, the load balancer will be configured to use port 80 and HTTP. • For **Production rule**, enter the **Evaluation order** and **Path pattern** for the rule. This rule is for your production (blue) service revision traffic. • For **Test rule**, enter the **Evaluation order** and **Path pattern** for the rule. This rule is for your test (green) service revision traffic.	

Elastic Load Balancing type	Steps	
	f. For **Target group**, configure the following: • For **Target group name**, enter a name and a protocol for the target group that the Application Load Balancer routes requests to. • For **Protocol**, choose the protocol for the target group that the Applicati on Load Balancer routes requests to. By default, the target group routes requests to the first container defined in your task definition. • For **Degregistration delay**, enter the number of seconds for the load balancer to change the target state to `UNUSED`. The default is 300 seconds. • For **Health check path**, enter an existing path within your container where the Application Load Balancer periodica	

Elastic Load Balancing type	Steps	
	lly sends requests to verify the connection health between the Application Load Balancer and the container. The default is the root directory (/). • For **Alternate group name**, enter the group name for the target group for your test (green) service revision.	

Elastic Load Balancing type	Steps	
Network Load Balancer	a. For **Load balancer type**, select **Network Load Balancer**. b. For **Load Balancer**, choose an existing Network Load Balancer. c. For **Choose container to load balance**, choose the container that hosts the service. d. For **Production listener**, choose the **Production listener port**, and the **Production listener protocol**. This is the listener for your production (blue) service revision traffic. e. For **Test listener**, choose the **Test listener port**, and the **Test listener protocol**. This is the listener for your test (green) service revision traffic. f. For **Target group**, configure the following: • For **Target group name**, enter a name and a protocol for the target group	

Elastic Load Balancing type	Steps	
	that the Network Load Balancer routes requests to. • For **Protocol**, choose the protocol for the target group that the Network Load Balancer routes requests to. By default, the target group routes requests to the first container defined in your task definition. • For **Degregistration delay**, enter the number of seconds for the load balancer to change the target state to `UNUSED`. The default is 300 seconds. • For **Health check path**, enter an existing path within your container where the Application Load Balancer periodica lly sends requests to verify the connectio n health between the Application Load Balancer and the container. The default	

Elastic Load Balancing type	Steps	
	is the root directory (/). • For **Alternate group name**, enter the group name for the target group for your test (green) service revision.	

11. (Optional) To help identify your service and tasks, expand the **Tags** section, and then configure your tags.

    To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the task definition tags, select **Turn on Amazon ECS managed tags**, and then for **Propagate tags from**, choose **Task definitions**.

    To have Amazon ECS automatically tag all newly launched tasks with the cluster name and the service tags, select **Turn on Amazon ECS managed tags**, and then for **Propagate tags from**, choose **Service**.

    Add or remove a tag.

    - [Add a tag] Choose **Add tag**, and then do the following:
      - For **Key**, enter the key name.
      - For **Value**, enter the key value.
    - [Remove a tag] Next to the tag, choose **Remove tag**.

12. Choose **Create**.

AWS CLI

1. Create a file named `service-definition.json` with the following content.

   Replace the *`user-input`* with your values.

```
{
```

```
  "serviceName": "myBlueGreenService",
  "cluster": "arn:aws:ecs:us-west-2:123456789012:cluster/sample-fargate-
cluster",
  "taskDefinition": "sample-fargate:1",
  "desiredCount": 5,
  "launchType": "FARGATE",
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "subnets": [
        "subnet-09ce6e74c116a2299",
        "subnet-00bb3bd7a73526788",
        "subnet-0048a611aaec65477"
      ],
      "securityGroups": [
        "sg-09d45005497daa123"
      ],
      "assignPublicIp": "ENABLED"
    }
  },
  "deploymentController": {
    "type": "ECS"
  },
  "deploymentConfiguration": {
    "strategy": "BLUE_GREEN",
    "maximumPercent": 200,
    "minimumHealthyPercent": 100,
    "bakeTimeInMinutes": 2,
    "alarms": {
      "alarmNames": [
        "myAlarm"
      ],
      "rollback": true,
      "enable": true
    },
    "lifecycleHooks": [
      {
        "hookTargetArn": "arn:aws:lambda:us-
west-2:7123456789012:function:checkExample",
        "roleArn": "arn:aws:iam::123456789012:role/ECSLifecycleHookInvoke",
        "lifecycleStages": [
          "PRE_SCALE_UP"
        ]
      }
    ]
```

```
  },
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/blue-target-group/54402ff563af1197",
      "containerName": "fargate-app",
      "containerPort": 80,
      "advancedConfiguration": {
        "alternateTargetGroupArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/green-target-group/cad10a56f5843199",
        "productionListenerRule": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/my-blue-green-
demo/32e0e4f946c3c05b/9cfa8c482e204f7d/831dbaf72edb911",
        "roleArn": "arn:aws:iam::123456789012:role/LoadBalancerManagementforECS"
      }
    }
  ]
}
```

2. Run `create-service`.

   Replace the *`user-input`* with your values.

```
aws ecs create-service --cli-input-json file://service-definition.json
```

   Alternatively, you can use the following example which creates a blue/green deployment service with a load balancer configuration:

```
aws ecs create-service \
   --cluster "arn:aws:ecs:us-west-2:123456789012:cluster/MyCluster" \
   --service-name "blue-green-example-service" \
   --task-definition "nginxServer:1" \
   --launch-type "FARGATE" \
   --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12345,subnet-67890,subnet-abcdef,subnet-
fedcba],securityGroups=[sg-12345],assignPublicIp=ENABLED}" \
   --desired-count 3 \
   --deployment-controller "type=ECS" \
   --deployment-configuration
 "strategy=BLUE_GREEN,maximumPercent=200,minimumHealthyPercent=100,bakeTimeInMinutes=0"
 \
   --load-balancers "targetGroupArn=arn:aws:elasticloadbalancing:us-
west-2:123456789012:targetgroup/MyBGtg1/
```

```
abcdef1234567890,containerName=nginx,containerPort=80,advancedConfiguration={alternateTa
west-2:123456789012:targetgroup/
MyBGtg2/0987654321fedcba,productionListenerRule=arn:aws:elasticloadbalancing:us-
west-2:123456789012:listener-rule/app/
MyLB/1234567890abcdef/1234567890abcdef,roleArn=arn:aws:iam::123456789012:role/
ELBManagementRole}"
```

Next steps

- Update the service to start the deployment. For more information, see Updating an Amazon ECS service.
- Monitor the deployment process to ensure it follows the blue/green pattern:
  - The green service revision is created and scaled up
  - Test traffic is routed to the green revision (if configured)
  - Production traffic is shifted to the green revision
  - After the bake time, the blue revision is terminated

Troubleshooting Amazon ECS blue/green deployments

This following provides solutions for common issues you might encounter when using blue/green deployments with Amazon ECS. Blue/green deployment errors can occur during the following phases:

- *Synchronous path*: Errors that appear immediately in response to `CreateService` or `UpdateService` API calls.
- *Asynchronous path*: Errors that appear in the `statusReason` field of `DescribeServiceDeployments` and cause a deployment rollback

Load balancer configuration issues

Load balancer configuration is a critical component of blue/green deployments in Amazon ECS. Proper configuration of listener rules, target groups, and load balancer types is essential for successful deployments. This section covers common load balancer configuration issues that can cause blue/green deployments to fail.

When troubleshooting load balancer issues, it's important to understand the relationship between listener rules and target groups. In a blue/green deployment:

- The production listener rule directs traffic to the currently active (blue) service revision
- The test listener rule can be used to validate the new (green) service revision before shifting production traffic
- Target groups are used to register the container instances from each service revision
- During deployment, traffic is gradually shifted from the blue service revision to the green service revision by adjusting the weights of the target groups in the listener rules

Listener rule configuration errors

The following issues relate to incorrect listener rule configuration for blue/green deployments.

Using an Application Load Balancer listener ARN instead of a listener rule ARN

Error message: `productionListenerRule has an invalid ARN format. Must be RuleArn for ALB or ListenerArn for NLB. Got: arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-alb/abc123/def456`

Solution: When using an Application Load Balancer, you must specify a listener rule ARN for `productionListenerRule` and `testListenerRule`, not a listener ARN. For Network Load Balancers, you must use the listener ARN.

For information about how to find the listener ARN, see [Listeners for your Application Load Balancers](#) in the *Application Load Balancer User Guide*. The ARN for a rule has the format *`arn:aws:elasticloadbalancing:region:account-id:listener-rule/app/...`*.

Using the same rule for both production and test listeners

Error message: `The following rules cannot be used as both production and test listener rules: arn:aws:elasticloadbalancing:us-west-2:123456789012:listener-rule/app/my-alb/abc123/def456/ghi789`

Solution: You must use different listener rules for production and test traffic. Create a separate listener rule for test traffic that routes to your test target group.

Missing test listener rule for a Network Load Balancer

Error message: `TestListenerRule is required for NLB with arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/nlb-targetgroup/abc123`

Solution: When you use a Network Load Balancer, you must specify both `productionListenerRule` and `testListenerRule`. Add a `testListenerRule` with a valid listener ARN to your configuration. For more information, see [Create a listener for your Network Load Balancer](#) in the *Network Load Balancer User Guide*

Target group not associated with listener rules

Error message: `Service deployment rolled back because of invalid networking configuration: Target group arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/myAlternateTG/abc123 is not associated with either productionListenerRule or testListenerRule.`

Solution: Both the primary target group and alternate target group must be associated with either the production listener rule or the test listener rule. Update your load balancer configuration to ensure both target groups are properly associated with your listener rules.

Missing test listener rule with an Application Load Balancer

Error message: `For Application LoadBalancer, testListenerRule is required when productionListenerRule is not associated with both targetGroup and alternateTargetGroup`

Solution: When you use an Application Load Balancer, if both target groups are not associated with the production listener rule, you must specify a test listener rule. Add a `testListenerRule` to your configuration and ensure both target groups are associated with either the production or test listener rule. For more information, see [Listeners for your Application Load Balancers](#) in the *Application Load Balancer User Guide*.

Target group configuration errors

The following issues relate to incorrect target group configuration for blue/green deployments.

Multiple target groups with traffic in listener rule

Error message: `Service deployment rolled back because of invalid networking configuration. productionListenerRule arn:aws:elasticloadbalancing:us-west-2:123456789012:listener-rule/app/my-alb/abc123/def456/ghi789 should have exactly one target group serving traffic but found 2 target groups which are serving traffic`

Solution: Before starting a blue/green deployment, ensure that only one target group is receiving traffic (has a non-zero weight) in your listener rule. Update your listener rule configuration to set the weight to zero for any target group that should not be receiving traffic.

Duplicate target groups across load balancer entries

Error message: `Duplicate targetGroupArn found: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/myecs-targetgroup/abc123`

Solution: Each target group ARN must be unique across all load balancer entries in your service definition. Review your configuration and ensure you're using different target groups for each load balancer entry.

Unexpected target group in production listener rule

Error message: `Service deployment rolled back because of invalid networking configuration. Production listener rule is forwarding traffic to unexpected target group arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/random-nlb-tg/abc123. Expected traffic to be forwarded to either targetGroupArn: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/nlb-targetgroup/def456 or alternateTargetGroupArn: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/nlb-tg-alternate/ghi789`

Solution: The production listener rule is forwarding traffic to a target group that is not specified in your service definition. Ensure that the listener rule is configured to forward traffic only to the target groups specified in your service definition.

For more information, see forward actions in the *Application Load Balancer User Guide*.

Load balancer type configuration errors

The following issues relate to incorrect load balancer type configuration for blue/green deployments.

Mixing Classic Load Balancer and Application Load Balancer or Network Load Balancer configurations

Error message: `All loadBalancers must be strictly either ELBv1 (defining loadBalancerName) or ELBv2 (defining targetGroupArn)`

> **Note**
>
> Classic Load Balancers are the previous generation of load balancers from Elastic Load Balancing. We recommend that you migrate to a current generation load balancer. For more information, see Migrate your Classic Load Balancer.

Solution: . Use either all Classic Load Balancers or all Application Load Balancers and Network Load Balancers.

For Application Load Balancers and Network Load Balancers, specify only the `targetGroupArn` field.

Using advanced configuration with a Classic Load Balancer

Error message: `advancedConfiguration field is not allowed with ELBv1 loadBalancers`

Solution: Advanced configuration for blue/green deployments is only supported with Application Load Balancers and Network Load Balancers. If you use a Classic Load Balancer (specified with `loadBalancerName`), you cannot use the `advancedConfiguration` field. Either switch to an Application Load Balancer, or remove the `advancedConfiguration` field.

Inconsistent advanced configuration across load balancers

Error message: `Either all or none of the provided loadBalancers must have advancedConfiguration defined`

Solution: If you're using multiple load balancers, you must either define `advancedConfiguration` for all of them or for none of them. Update your configuration to ensure consistency across all load balancer entries.

Missing advanced configuration with blue/green deployment

Error message: `advancedConfiguration field is required for all loadBalancers when using a non-ROLLING deployment strategy`

Solution: When using a blue/green deployment strategy with Application Load Balancers, you must specify the `advancedConfiguration` field for all load balancer entries. Add the required `advancedConfiguration` to your load balancer configuration.

Permission issues

The following issues relate to insufficient permissions for blue/green deployments.

Missing trust policy on infrastructure role

Error message: `Service deployment rolled back because of invalid networking configuration. ECS was unable to manage the ELB resources due to missing permissions on ECS Infrastructure Role 'arn:aws:iam::123456789012:role/Admin'.`

Solution: The IAM role specified for managing load balancer resources does not have the correct trust policy. Update the role's trust policy to allow the service to assume the role. The trust policy must include:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Missing read permissions on load balancer role

Error message: `service myService failed to describe target health on target-group myTargetGroup with (error User: arn:aws:sts::123456789012:assumed-role/myELBRole/ecs-service-scheduler is not authorized to perform: elasticloadbalancing:DescribeTargetHealth because no identity-based policy allows the elasticloadbalancing:DescribeTargetHealth action)`

Solution: The IAM role used for managing load balancer resources does not have permission to read target health information. Add the `elasticloadbalancing:DescribeTargetHealth` permission to the role's policy. For information about Elastic Load Balancing permissions, see Amazon ECS infrastructure IAM role for load balancers.

Missing write permissions on load balancer role

Error message: `service myService failed to register targets in target-group myTargetGroup with (error User: arn:aws:sts::123456789012:assumed-role/myELBRole/ecs-service-scheduler is not authorized to perform: elasticloadbalancing:RegisterTargets on resource: arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/myTargetGroup/abc123 because no identity-based policy allows the elasticloadbalancing:RegisterTargets action)`

Solution: The IAM role used for managing load balancer resources does not have permission to register targets. Add the `elasticloadbalancing:RegisterTargets` permission to the role's policy. For information about Elastic Load Balancing permissions, see Amazon ECS infrastructure IAM role for load balancers.

Missing permission to modify listener rules

Error message: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. User: arn:aws:sts::123456789012:assumed-role/myELBRole/ECSNetworkingWithELB is not authorized to perform: elasticloadbalancing:ModifyListener on resource: arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-alb/abc123/def456 because no identity-based policy allows the elasticloadbalancing:ModifyListener action`

Solution: The IAM role used for managing load balancer resources does not have permission to modify listeners. Add the `elasticloadbalancing:ModifyListener` permission to the role's policy. For information about Elastic Load Balancing permissions, see Amazon ECS infrastructure IAM role for load balancers.

For blue/green deployments, we recommend attaching the `AmazonECS-ServiceLinkedRolePolicy` managed policy to your infrastructure role, which includes all the necessary permissions for managing load balancer resources.

Lifecycle hook issues

The following issues relate to lifecycle hooks in blue/green deployments.

Incorrect trust policy on Lambda hook role

Error message: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. ECS was unable to assume role arn:aws:iam::123456789012:role/Admin`

Solution: The IAM role specified for the Lambda lifecycle hook does not have the correct trust policy. Update the role's trust policy to allow the service to assume the role. The trust policy must include:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Lambda hook returns FAILED status

Error message: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. Lifecycle hook target arn:aws:lambda:us-west-2:123456789012:function:myHook returned FAILED status.`

Solution: The Lambda function specified as a lifecycle hook returned a FAILED status. Check the Lambda function logs in Amazon CloudWatch logs to determine the failure reason, and update the function to handle the deployment event correctly.

Missing permission to invoke Lambda function

Error message: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. ECS was unable to invoke hook target arn:aws:lambda:us-west-2:123456789012:function:myHook due to User:`

`arn:aws:sts::123456789012:assumed-role/myLambdaRole/ECS-Lambda-Execution is not authorized to perform: lambda:InvokeFunction on resource: arn:aws:lambda:us-west-2:123456789012:function:myHook because no identity-based policy allows the lambda:InvokeFunction action`

Solution: The IAM role used for the Lambda lifecycle hook does not have permission to invoke the Lambda function. Add the `lambda:InvokeFunction` permission to the role's policy for the specific Lambda function ARN. For information about Lambda permissions, see Permissions required for Lambda functions in Amazon ECS blue/green deployments.

Lambda function timeout or invalid response

Error message: `Service deployment rolled back because TEST_TRAFFIC_SHIFT lifecycle hook(s) failed. ECS was unable to parse the response from arn:aws:lambda:us-west-2:123456789012:function:myHook due to HookStatus must not be null`

Solution: The Lambda function either timed out or returned an invalid response. Ensure that your Lambda function returns a valid response with a `hookStatus` field set to either `SUCCEEDED` or `FAILED`. Also, check that the Lambda function timeout is set appropriately for your validation logic. For more information, see Lifecycle hooks for Amazon ECS service deployments.

Example of a valid Lambda response:

```
{
  "hookStatus": "SUCCEEDED",
  "reason": "Validation passed"
}
```

Deploy Amazon ECS services using a third-party controller

The *external* deployment type allows you to use any third-party deployment controller for full control over the deployment process for an Amazon ECS service. The details for your service are managed by either the service management API actions (`CreateService`, `UpdateService`, and `DeleteService`) or the task set management API actions (`CreateTaskSet`, `UpdateTaskSet`, `UpdateServicePrimaryTaskSet`, and `DeleteTaskSet`). Each API action manages a subset of the service definition parameters.

The `UpdateService` API action updates the desired count and health check grace period parameters for a service. If the launch type, platform version, load balancer details, network configuration, or task definition need to be updated, you must create a new task set.

The `UpdateTaskSet` API action updates only the scale parameter for a task set.

The `UpdateServicePrimaryTaskSet` API action modifies which task set in a service is the primary task set. When you call the `DescribeServices` API action, it returns all fields specified for a primary task set. If the primary task set for a service is updated, any task set parameter values that exist on the new primary task set that differ from the old primary task set in a service are updated to the new value when a new primary task set is defined. If no primary task set is defined for a service, when describing the service, the task set fields are null.

External deployment considerations

Consider the following when using the external deployment type:

- The supported load balancer types are either an Application Load Balancer or a Network Load Balancer.
- The Fargate launch type or `EXTERNAL` deployment controller types don't support the `DAEMON` scheduling strategy.
- Application AutoScaling integration with Amazon ECS only supports Amazon ECS service as a target. The supported scalable dimension for Amazon ECS is `ecs:service:DesiredCount` - the task count of an Amazon ECS service. There is no direct integration between Application AutoScaling and Amazon ECS task sets. Amazon ECS task sets calculate the `ComputedDesiredCount` based on the Amazon ECS service `DesiredCount`.

External deployment workflow

The following is the basic workflow for managing an external deployment on Amazon ECS.

To manage an Amazon ECS service using an external deployment controller

1. Create an Amazon ECS service. The only required parameter is the service name. You can specify the following parameters when creating a service using an external deployment controller. All other service parameters are specified when creating a task set within the service.

`serviceName`

Type: String

Required: Yes

The name of your service. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a Region or across multiple Regions.

`desiredCount`

The number of instantiations of the specified task set task definition to place and keep running within the service.

`deploymentConfiguration`

Optional deployment parameters that control how many tasks run during a deployment and the ordering of stopping and starting tasks.

`tags`

Type: Array of objects

Required: No

The metadata that you apply to the service to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see [Tagging Amazon ECS resources](#).

When you update a service, this parameter doesn't trigger a new service deployment.

`key`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

`value`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

`enableECSManagedTags`

Specifies whether to use Amazon ECS managed tags for the tasks within the service. For more information, see [Use tags for billing](#).

`propagateTags`

Type: String

Valid values: `TASK_DEFINITION` | `SERVICE`

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation or task creation, use the `TagResource` API action.

When you update a service, this parameter doesn't trigger a new service deployment.

`schedulingStrategy`

The scheduling strategy to use. Services using an external deployment controller support only the `REPLICA` scheduling strategy.

`placementConstraints`

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task (this limit includes constraints in the task definition and those specified at run time). If you are using the Fargate launch type, task placement constraints aren't supported.

`placementStrategy`

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

The following is an example service definition for creating a service using an external deployment controller.

```
{
    "cluster": "",
    "serviceName": "",
    "desiredCount": 0,
    "role": "",
    "deploymentConfiguration": {
        "maximumPercent": 0,
        "minimumHealthyPercent": 0
    },
    "placementConstraints": [
        {
            "type": "distinctInstance",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
```

```
    "propagateTags": "TASK_DEFINITION"
}
```

2. Create an initial task set. The task set contains the following details about your service:

   `taskDefinition`

      The task definition for the tasks in the task set to use.

   `launchType`

      Type: String

      Valid values: `EC2` | `FARGATE` | `EXTERNAL`

      Required: No

      The launch type on which to run your service. If a launch type is not specified, the default `capacityProviderStrategy` is used by default. For more information, see [Amazon ECS launch types](#).

      When you update a service, this parameter triggers a new service deployment.

      If a `launchType` is specified, the `capacityProviderStrategy` parameter must be omitted.

   `platformVersion`

      Type: String

      Required: No

      The platform version on which your tasks in the service are running. A platform version is only specified for tasks using the Fargate launch type. If one is not specified, the latest version (`LATEST`) is used by default.

      When you update a service, this parameter triggers a new service deployment.

      AWS Fargate platform versions are used to refer to a specific runtime environment for the Fargate task infrastructure. When specifying the `LATEST` platform version when running a task or creating a service, you get the most current platform version available for your tasks. When you scale up your service, those tasks receive the platform version that was specified on the service's current deployment. For more information, see [Fargate platform versions for Amazon ECS](#).

> **Note**
>
> Platform versions are not specified for tasks using the EC2 launch type.

`loadBalancers`

A load balancer object representing the load balancer to use with your service. When using an external deployment controller, only Application Load Balancers and Network Load Balancers are supported. If you're using an Application Load Balancer, only one Application Load Balancer target group is allowed per task set.

The following snippet shows an example `loadBalancer` object to use.

```
"loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
]
```

> **Note**
>
> When specifying a `loadBalancer` object, you must specify a `targetGroupArn` and omit the `loadBalancerName` parameters.

`networkConfiguration`

The network configuration for the service. This parameter is required for task definitions that use the `awsvpc` network mode to receive their own elastic network interface, and it's not supported for other network modes. For more information about networking for the Fargate launch type, see [Amazon ECS task networking options for the Fargate launch type](#).

`serviceRegistries`

The details of the service discovery registries to assign to this service. For more information, see Use service discovery to connect Amazon ECS services with DNS names.

`scale`

A floating-point percentage of the desired number of tasks to place and keep running in the task set. The value is specified as a percent total of a service's `desiredCount`. Accepted values are numbers between 0 and 100.

The following is a JSON example for creating a task set for an external deployment controller.

```
{
    "service": "",
    "cluster": "",
    "externalId": "",
    "taskDefinition": "",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
```

```
        }
    ],
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "scale": {
        "value": null,
        "unit": "PERCENT"
    },
    "clientToken": ""
}
```

3. When service changes are needed, use the `UpdateService`, `UpdateTaskSet`, or `CreateTaskSet` API action depending on which parameters you're updating. If you created a task set, use the `scale` parameter for each task set in a service to determine how many tasks to keep running in the service. For example, if you have a service that contains `tasksetA` and you create a `tasksetB`, you might test the validity of `tasksetB` before wanting to transition production traffic to it. You could set the `scale` for both task sets to `100`, and when you were ready to transition all production traffic to `tasksetB`, you could update the `scale` for `tasksetA` to `0` to scale it down.

CodeDeploy blue/green deployments for Amazon ECS

We recommend that you use the Amazon ECS blue/green deployment. For more information, see [Creating an Amazon ECS blue/green deployment](#).

The *blue/green* deployment type uses the blue/green deployment model controlled by CodeDeploy. Use this deployment type to verify a new deployment of a service before sending production traffic to it. For more information, see [What Is CodeDeploy](#) in the *AWS CodeDeploy User Guide*.Validate the state of an Amazon ECS service before deployment

There are three ways traffic can shift during a blue/green deployment:

- **Canary** — Traffic is shifted in two increments. You can choose from predefined canary options that specify the percentage of traffic shifted to your updated task set in the first increment and the interval, in minutes, before the remaining traffic is shifted in the second increment.
- **Linear** — Traffic is shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment.
- **All-at-once** — All traffic is shifted from the original task set to the updated task set all at once.

The following are components of CodeDeploy that Amazon ECS uses when a service uses the blue/green deployment type:

CodeDeploy application

A collection of CodeDeploy resources. This consists of one or more deployment groups.

CodeDeploy deployment group

The deployment settings. This consists of the following:

- Amazon ECS cluster and service
- Load balancer target group and listener information
- Deployment roll back strategy
- Traffic rerouting settings
- Original revision termination settings
- Deployment configuration
- CloudWatch alarms configuration that can be set up to stop deployments
- SNS or CloudWatch Events settings for notifications

For more information, see Working with Deployment Groups in the *AWS CodeDeploy User Guide*.

CodeDeploy deployment configuration

Specifies how CodeDeploy routes production traffic to your replacement task set during a deployment. The following pre-defined linear and canary deployment configuration are available. You can also create custom defined linear and canary deployments as well. For more information, see Working with Deployment Configurations in the *AWS CodeDeploy User Guide*.

- **CodeDeployDefault.ECSAllAtOnce**: Shifts all traffic to the updated Amazon ECS container at once

- **CodeDeployDefault.ECSLinear10PercentEvery1Minutes**: Shifts 10 percent of traffic every minute until all traffic is shifted.
- **CodeDeployDefault.ECSLinear10PercentEvery3Minutes**: Shifts 10 percent of traffic every 3 minutes until all traffic is shifted.
- **CodeDeployDefault.ECSCanary10Percent5Minutes**: Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed five minutes later.
- **CodeDeployDefault.ECSCanary10Percent15Minutes**: Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed 15 minutes later.

Revision

A revision is the CodeDeploy application specification file (AppSpec file). In the AppSpec file, you specify the full ARN of the task definition and the container and port of your replacement task set where traffic is to be routed when a new deployment is created. The container name must be one of the container names referenced in your task definition. If the network configuration or platform version has been updated in the service definition, you must also specify those details in the AppSpec file. You can also specify the Lambda functions to run during the deployment lifecycle events. The Lambda functions allow you to run tests and return metrics during the deployment. For more information, see AppSpec File Reference in the *AWS CodeDeploy User Guide*.

Considerations

Consider the following when using the blue/green deployment type:

- When an Amazon ECS service using the blue/green deployment type is initially created, an Amazon ECS task set is created.
- You must configure the service to use either an Application Load Balancer or Network Load Balancer. The following are the load balancer requirements:
  - You must add a production listener to the load balancer, which is used to route production traffic.
  - An optional test listener can be added to the load balancer, which is used to route test traffic. If you specify a test listener, CodeDeploy routes your test traffic to the replacement task set during a deployment.
  - Both the production and test listeners must belong to the same load balancer.
  - You must define a target group for the load balancer. The target group routes traffic to the original task set in a service through the production listener.

  - When a Network Load Balancer is used, only the `CodeDeployDefault.ECSAllAtOnce` deployment configuration is supported.
- For services configured to use service auto scaling and the blue/green deployment type, auto scaling is not blocked during a deployment but the deployment may fail under some circumstances. The following describes this behavior in more detail.
  - If a service is scaling and a deployment starts, the green task set is created and CodeDeploy will wait up to an hour for the green task set to reach steady state and won't shift any traffic until it does.
  - If a service is in the process of a blue/green deployment and a scaling event occurs, traffic will continue to shift for 5 minutes. If the service doesn't reach steady state within 5 minutes, CodeDeploy will stop the deployment and mark it as failed.
- Tasks using the Fargate launch type or the `CODE_DEPLOY` deployment controller types don't support the `DAEMON` scheduling strategy.
- When you initially create a CodeDeploy application and deployment group, you must specify the following:
  - You must define two target groups for the load balancer. One target group should be the initial target group defined for the load balancer when the Amazon ECS service was created. The second target group's only requirement is that it can't be associated with a different load balancer than the one the service uses.
- When you create a CodeDeploy deployment for an Amazon ECS service, CodeDeploy creates a *replacement task set* (or *green task set*) in the deployment. If you added a test listener to the load balancer, CodeDeploy routes your test traffic to the replacement task set. This is when you can run any validation tests. Then CodeDeploy reroutes the production traffic from the original task set to the replacement task set according to the traffic rerouting settings for the deployment group.

Required IAM permissions

Blue/green deployments are made possible by a combination of the Amazon ECS and CodeDeploy APIs. Users must have the appropriate permissions for these services before they can use Amazon ECS blue/green deployments in the AWS Management Console or with the AWS CLI or SDKs.

In addition to the standard IAM permissions for creating and updating services, Amazon ECS requires the following permissions. These permissions have been added to the `AmazonECS_FullAccess` IAM policy. For more information, see AmazonECS_FullAccess.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:CreateApplication",
                "codedeploy:CreateDeployment",
                "codedeploy:CreateDeploymentGroup",
                "codedeploy:GetApplication",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentGroup",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "codedeploy:ListDeployments",
                "codedeploy:StopDeployment",
                "codedeploy:GetDeploymentTarget",
                "codedeploy:ListDeploymentTargets",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:GetApplicationRevision",
                "codedeploy:RegisterApplicationRevision",
                "codedeploy:BatchGetApplicationRevisions",
                "codedeploy:BatchGetDeploymentGroups",
                "codedeploy:BatchGetDeployments",
                "codedeploy:BatchGetApplications",
                "codedeploy:ListApplicationRevisions",
                "codedeploy:ListDeploymentConfigs",
                "codedeploy:ContinueDeployment",
                "sns:ListTopics",
                "cloudwatch:DescribeAlarms",
                "lambda:ListFunctions"
            ],
            "Resource": ["*"]
        }
    ]
}
```

Note

In addition to the standard Amazon ECS permissions required to run tasks and services, users also require `iam:PassRole` permissions to use IAM roles for tasks.

CodeDeploy needs permissions to call Amazon ECS APIs, modify your Elastic Load Balancing, invoke Lambda functions, and describe CloudWatch alarms, as well as permissions to modify your service's desired count on your behalf. Before creating an Amazon ECS service that uses the blue/green deployment type, you must create an IAM role (`ecsCodeDeployRole`). For more information, see Amazon ECS CodeDeploy IAM Role.

The Create Amazon ECS service example and Update Amazon ECS service example IAM policy examples show the permissions that are required for users to use Amazon ECS blue/green deployments on the AWS Management Console.

Migrate CodeDeploy blue/green deployments to Amazon ECS blue/green deployments

CodeDeploy blue/green and Amazon ECS blue/green deployments provide similar functionality, but they differ in how you configure and manage them.

CodeDeploy blue/green deployment overview

When creating an Amazon ECS service using CodeDeploy, you:

1. Create a load balancer with a production listener and (optionally) a test listener. Each listener is configured with a single (default) rule that routes all traffic to a single target group (the primary target group).
2. Create an Amazon ECS service, configured to use the listener and target group, with `deploymentController` type set to `CODE_DEPLOY`. Service creation results in the creation of a (blue) task set registered with the specified target group.
3. Create a CodeDeploy deployment group (as part of a CodeDeploy application), and configure it with details of the Amazon ECS cluster, service name, load balancer listeners, two target groups (the primary target group used in the production listener rule, and a secondary target group to be used for replacement tasks), a service role (to grant CodeDeploy permissions to manipulate Amazon ECS and Elastic Load Balancing resources) and various parameters that control the deployment behavior.

With CodeDeploy, new versions of a service are deployed using `CreateDeployment()`, specifying the CodeDeploy application name, deployment group name, and an AppSpec file which provides details of the new revision and optional lifecycle hooks. The CodeDeploy deployment creates a replacement (green) task set and registers its tasks with the secondary target group. When this becomes healthy, it is available for testing (optional) and for production. In both cases, re-routing is achieved by changing the respective listener rule to point at the secondary target group associated with the green task set. Rollback is achieved by changing the production listener rule back to the primary target group.

Amazon ECS blue/green deployment overview

With Amazon ECS blue/green deployments, The deployment configuration is part of the Amazon ECS service itself:

1. You must pre-configure the load balancer production listener with a rule that includes two target groups with weights of 1 and 0.
2. You need to specify the following resources, or update the service resources:
   - The ARN of this listener rule
   - The two target groups
   - An IAM role to grant Amazon ECS permission to call the Elastic Load Balancing APIs
   - An optional IAM role to run Lambda functions
   - Set `deploymentController` type to `ECS` and `deploymentConfiguration.strategy` to `BLUE_GREEN`. This results in the creation of a (blue) service deployment whose tasks are registered with the primary target group.

With Amazon ECS blue/green, a new service revision is created by calling Amazon ECS `UpdateService()`, passing details of the new revision. The service deployment creates new (green) service revision tasks and registers them with the secondary target group. Amazon ECS handles re-routing and rollback operations by switching the weights on the listener rule.

Key implementation differences

While both approaches result in the creation of an initial set of tasks, the underlying implementation differs:

- CodeDeploy uses a task set, whereas Amazon ECS uses a service revision. Amazon ECS task sets are an older construct which have been superseded by Amazon ECS service revisions and deployments. The latter offer greater visibility into the deployment process, as well as the service deployment and service revision history.
- With CodeDeploy, lifecycle hooks are specified as part of the AppSpec file that is supplied to `CreateDeployment()`. This means that the hooks can be changed from one deployment to the next. With Amazon ECS blue/green, the hooks are specified as part of the service configuration, and any updates would require an `UpdateService()` call.

- Both CodeDeploy and Amazon ECS blue/green use Lambda for hook implementation, but the expected inputs and outputs differ.

  With CodeDeploy, the function must call `PutLifecycleEventHookExecutionStatus()` to return the hook status, which can either be `SUCCEEDED` or `FAILED`. With Amazon ECS, the Lambda response is used to indicate the hook status.
- CodeDeploy invokes each hook as a one-off call, and expects a final execution status to be returned within one hour. Amazon ECS hooks are more flexible in that they can return an `IN_PROGRESS` indicator, which signals that the hook must be re-invoked repeatedly until it results in `SUCCEEDED` or `FAILED`. For more information, see Lifecycle hooks for Amazon ECS service deployments.

Migration approaches

There are three main approaches to migrating from CodeDeploy blue/green to Amazon ECS blue/green deployments. Each approach has different characteristics in terms of complexity, risk, rollback capability, and potential downtime.

Reuse the same Elastic Load Balancing resources used for CodeDeploy

You update the existing Amazon ECS service to use the Amazon ECS deployment controller with blue/green deployment strategy instead of CodeDeploy deployment controller. Consider the following when using this approach:

- The migration procedure is simpler because you are updating the existing Amazon ECS service deployment controller and deployment strategy.
- There is no downtime when correctly configured and migrated.
- A rollback requires that you revert the service revision.
- The risk is high because there is no parallel blue/green configuration.

You use the same load balancer listener and target groups that are used for CodeDeploy. If you are using AWS CloudFormation, see Migrating an AWS CloudFormation CodeDeploy blue/green deployment template to an Amazon ECS blue/green AWS CloudFormation template.

1. Modify the default rule of the production/test listeners to include the alternate target group and set the weight of the primary target group to 1 and the alternate target group to 0.

For CodeDeploy, the listeners of the load balancer attached to the service are configured with a single (default) rule that routes all traffic to a single target group. For Amazon ECS blue/green, the load balancer listeners must be pre-configured with a rule that includes the two target groups with weights. The primary target group must be weighted to 1 and the alternate target group must be weighted to 0.

2. Update the existing Amazon ECS service by calling the `UpdateService` API and setting the parameter `deploymentController` to `ECS`, and the parameter `deploymentStrategy` to `BLUE_GREEN`. Specify the ARNs of the target group, the alternative target group, the production listener, and an optional test listener.
3. Verify that the service works as expected.
4. Delete the CodeDeploy setup for this Amazon ECS service as you are now using Amazon ECS blue/green.

New service with existing load balancer

This approach uses the blue/green strategy for the migration.

Consider the following when using this approach:

- There is minimal disruption. It occurs only during the Elastic Load Balancing port swap.
- A rollback requires that you revert the Elastic Load Balancing port swap.
- The risk is low because there are parallel configurations. Therefore you can test before the traffic shift.

1. Leave the listeners, the target groups, and the Amazon ECS service for the CodeDeploy setup intact so you can easily rollback to this setup if needed.
2. Create new target groups and new listeners (with different ports from the original listeners) under the existing load balancer. Then, create a new Amazon ECS service that matches the existing Amazon ECS service except that you use `ECS` as deployment controller, `BLUE_GREEN` as a deployment strategy, and pass the ARNs for the new target groups and the new listeners rules.
3. Verify the new setup by manually sending HTTP traffic to the service. If everything goes well, swap the ports of the original listeners and the new listeners to route the traffic to the new setup.

4. Verify the new setup, and if everything continues to work as expected, delete the CodeDeploy setup.

New service with a new load balancer

Like the previous approach, this approach uses the blue/green strategy for the migration. The key difference is that the switch from the CodeDeploy setup to the Amazon ECS blue/green setup happens at a reverse proxy layer above the load balancer. Sample implementations for the reverse proxy layer are Route 53 and CloudFront.

This approach is suitable for customers who already have this reverse proxy layer, and if all the communication with the service is happening through it (for example, no direct communication at the load balancer level).

Consider the following when using this approach:

- This requires a reverse proxy layer.
- The migration procedure is more complex because you need to update the existing Amazon ECS service deployment controller and deployment strategy.
- There is minimal disruption. It occurs only during the Elastic Load Balancing port swap.
- A rollback requires that you reverse the proxy configuration changes.
- The risk is low because there are parallel configurations. Therefore you can test before the traffic shift.

1. Do not modify the existing CodeDeploy setup intact (load balancer, listeners, target groups, Amazon ECS service, and CodeDeploy deployment group).
2. Create a new load balancer, target groups, and listeners configured for Amazon ECS blue/green deployments.

   Configure the appropriate resources.

   - Application Load Balancer - For more information, see Application Load Balancer resources for blue/green deployments.
   - Network Load Balancer - For more information, see Network Load Balancer resources for Amazon ECS blue/green deployments.
3. Create a new service with ECS as deployment controller and `BLUE_GREEN` as deployment strategy, pointing to the new load balancer resources.

4. Verify the new setup by testing it through the new load balancer.
5. Update the reverse proxy configuration to route traffic to the new load balancer.
6. Observe the new service revision, and if everything continues to work as expected, delete the CodeDeploy setup.

Next steps

After migrating to Amazon ECS blue/green deployments:

- Update your deployment scripts and CI/CD pipelines to use the Amazon ECS `UpdateService` API instead of the CodeDeploy `CreateDeployment` API.
- Update your monitoring and alerting to track Amazon ECS service deployments instead of CodeDeploy deployments.
- Consider implementing automated testing of your new deployment process to ensure it works as expected.

Migrating from a CodeDeploy blue/green to an Amazon ECS blue/green service deployment

By using Amazon ECS blue/green deployments, you can make and test service changes before implementing them in a production environment.

You must create new lifecycle hooks for your Amazon ECS blue/green deployment.

Prerequisites

Perform the following operations before you start a blue/green deployment.

1. Replace the Amazon ECS CodeDeploy IAM role with the following permissions.
   - For information about Elastic Load Balancing permissions, see Amazon ECS infrastructure IAM role for load balancers.
   - For information about Lambda permissions, see Permissions required for Lambda functions in Amazon ECS blue/green deployments.
2. Turn off CodeDeploy automation. For more information, see Working with deployment groups in CodeDeploy in the *CodeDeploy User Guide*.
3. Make sure that you have the following information from your CodeDeploy blue/green deployment. You can reuse this information for the Amazon ECS blue/green deployment:
   - The production target group

- The production listener
- The production rule
- The test target group

  This is the target group for the green service revision,

4. Ensure that your Application Load Balancer target groups are properly associated with listener rules:
   - If you are not using test listeners, both target groups (production and test) must be associated with production listener rules.
   - If you are using test listeners, one target group must be linked to production listener rules and the other target group must be linked to test listener rules.

   If this requirement is not met, the service deployment will fail with the following error: `Service deployment rolled back because of invalid networking configuration. Both targetGroup and alternateTargetGroup must be associated with the productionListenerRule or testListenerRule.`
5. Verify that there are no ongoing service deployments for the service. For more information, see [View service history using Amazon ECS service deployments].
6. Amazon ECS blue/green deployments require your service to use one of the following features: Configure the appropriate resources.
   - Application Load Balancer - For more information, see [Application Load Balancer resources for blue/green deployments].
   - Network Load Balancer - For more information, see [Network Load Balancer resources for Amazon ECS blue/green deployments].
   - Service Connect - For more information, see [Service Connect resources for Amazon ECS blue/green deployments].
7. Decide if you want to run Lambda functions for the lifecycle stages for the stages in the Amazon ECS blue/green deployment.
   - Pre scale up
   - After scale up
   - Test traffic shift
   - After test traffic shift
   - Production traffic shift

- After production traffic shift

  Create Lambda functions for each lifecycle stage. For more information, see Create a Lambda function with the console in the *AWS Lambda Developer Guide*.

For more information about updating a service's deployment controller, see Update Amazon ECS service parameters.

Procedure

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.

   The cluster details page displays.
3. From the **Services** tab, choose the service.

   The service details page displays.
4. In the banner, choose **Update deployment controller type**.

   The **Migrate deployment controller type** page displays.
5. Expand **New**, and then specify the following parameters.

   a. For **Deployment controller type**, choose **ECS**.

   b. For **Deployment strategy**, choose **Blue/green**.

   c. For **Bake time**, enter the time that both the blue and green service revisions run.

   d. To run Lambda functions for a lifecycle stage, under **Deployment lifecyce hooks** do the following for each unique Lambda function:

      i. Choose **Add**.

         Repeat for every unique function you want to run.

      ii. For **Lambda function**, enter the function name.

      iii. For **Role**, choose the role that you created in the prerequisites with the blue/green permissions.

         For more information, see Permissions required for Lambda functions in Amazon ECS blue/green deployments.

iv. For **Lifecycle stages**, select the stages the Lambda function runs.

v. (Optional) For **Hook details**, enter a key-value pair that provides information about the hook.

6. Expand **Load Balancing**, and the configure the following:

   a. For **Role**, choose the role that you created in the prerequisites with the blue/green permissions.

      For more information, see Permissions required for Lambda functions in Amazon ECS blue/green deployments.

   b. For **Listener**, choose the production listener from your CodeDeploy blue/green deployment.

   c. For **Production rule**, choose the production rule from your CodeDeploy blue/green deployment.

   d. For **Test rule**, choose the test rule from your CodeDeploy blue/green deployment.

   e. For **Target group**, choose the production target group from your CodeDeploy blue/green deployment.

   f. For **Alternate target group**, choose the test target group from your CodeDeploy blue/green deployment.

7. Choose **Update**.

Next steps

- Update the service to start the deployment. For more information, see Updating an Amazon ECS service.
- Monitor the deployment process to ensure it follows the blue/green pattern:
  - The green service revision is created and scaled up
  - Test traffic is routed to the green revision (if configured)
  - Production traffic is shifted to the green revision
  - After the bake time, the blue revision is terminated

Migrating an AWS CloudFormation CodeDeploy blue/green deployment template to an Amazon ECS blue/green AWS CloudFormation template

Migrate a AWS CloudFormation template that uses CodeDeploy blue/green deployments for Amazon ECS services to one that uses the native Amazon ECS blue/green deployment strategy. The migration follows the "Reuse the same Elastic Load Balancing resources used for CodeDeploy" approach. For more information, see the section called “Migrate CodeDeploy blue/green deployments to Amazon ECS blue/green deployments” documentation.

Source template

This template uses the `AWS::CodeDeployBlueGreen` transform and `AWS::CodeDeploy::BlueGreen` hook to implement blue/green deployments for an Amazon ECS service.

Source

This is the complete AWS CloudFormation template using CodeDeploy blue/green deployment. For more information, see Blue/green deployment template example in the *AWS CloudFormation User Guide*:

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "Vpc": {
      "Type": "AWS::EC2::VPC::Id"
    },
    "Subnet1": {
      "Type": "AWS::EC2::Subnet::Id"
    },
    "Subnet2": {
      "Type": "AWS::EC2::Subnet::Id"
    }
  },
  "Transform": [
    "AWS::CodeDeployBlueGreen"
  ],
  "Hooks": {
    "CodeDeployBlueGreenHook": {
      "Type": "AWS::CodeDeploy::BlueGreen",
      "Properties": {
        "TrafficRoutingConfig": {
          "Type": "TimeBasedCanary",
```

```
          "TimeBasedCanary": {
            "StepPercentage": 15,
            "BakeTimeMins": 5
          }
        },
        "Applications": [
          {
            "Target": {
              "Type": "AWS::ECS::Service",
              "LogicalID": "ECSDemoService"
            },
            "ECSAttributes": {
              "TaskDefinitions": [
                "BlueTaskDefinition",
                "GreenTaskDefinition"
              ],
              "TaskSets": [
                "BlueTaskSet",
                "GreenTaskSet"
              ],
              "TrafficRouting": {
                "ProdTrafficRoute": {
                  "Type": "AWS::ElasticLoadBalancingV2::Listener",
                  "LogicalID": "ALBListenerProdTraffic"
                },
                "TargetGroups": [
                  "ALBTargetGroupBlue",
                  "ALBTargetGroupGreen"
                ]
              }
            }
          }
        ]
      }
    }
  },
  "Resources": {
    "ExampleSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Security group for ec2 access",
        "VpcId": {"Ref": "Vpc"},
        "SecurityGroupIngress": [
          {
```

```
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 8080,
            "ToPort": 8080,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "0.0.0.0/0"
          }
        ]
      }
    },
    "ALBTargetGroupBlue": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
```

```
    },
    "ALBTargetGroupGreen": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ExampleALB": {
      "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
      "Properties": {
        "Scheme": "internet-facing",
        "SecurityGroups": [
          {"Ref": "ExampleSecurityGroup"}
        ],
        "Subnets": [
          {"Ref": "Subnet1"},
          {"Ref": "Subnet2"}
        ],
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "Type": "application",
```

```
        "IpAddressType": "ipv4"
      }
    },
    "ALBListenerProdTraffic": {
      "Type": "AWS::ElasticLoadBalancingV2::Listener",
      "Properties": {
        "DefaultActions": [
          {
            "Type": "forward",
            "ForwardConfig": {
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                }
              ]
            }
          }
        ],
        "LoadBalancerArn": {"Ref": "ExampleALB"},
        "Port": 80,
        "Protocol": "HTTP"
      }
    },
    "ALBListenerProdRule": {
      "Type": "AWS::ElasticLoadBalancingV2::ListenerRule",
      "Properties": {
        "Actions": [
          {
            "Type": "forward",
            "ForwardConfig": {
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                }
              ]
            }
          }
        ],
        "Conditions": [
          {
            "Field": "http-header",
            "HttpHeaderConfig": {
```

```
              "HttpHeaderName": "User-Agent",
              "Values": [
                "Mozilla"
              ]
            }
          }
        ],
        "ListenerArn": {"Ref": "ALBListenerProdTraffic"},
        "Priority": 1
      }
    },
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "",
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    },
    "BlueTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "ContainerDefinitions": [
          {
            "Name": "DemoApp",
            "Image": "nginxdemos/hello:latest",
            "Essential": true,
            "PortMappings": [
              {
                "HostPort": 80,
```

```
                "Protocol": "tcp",
                "ContainerPort": 80
              }
            ]
          }
        ],
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "NetworkMode": "awsvpc",
        "Cpu": "256",
        "Memory": "512",
        "Family": "ecs-demo"
      }
    },
    "ECSDemoCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {}
    },
    "ECSDemoService": {
      "Type": "AWS::ECS::Service",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "DesiredCount": 1,
        "DeploymentController": {
          "Type": "EXTERNAL"
        }
      }
    },
    "BlueTaskSet": {
      "Type": "AWS::ECS::TaskSet",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "LaunchType": "FARGATE",
        "NetworkConfiguration": {
          "AwsVpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": [
              {"Ref": "ExampleSecurityGroup"}
            ],
            "Subnets": [
              {"Ref": "Subnet1"},
              {"Ref": "Subnet2"}
            ]
```

```
          }
        },
        "PlatformVersion": "1.4.0",
        "Scale": {
          "Unit": "PERCENT",
          "Value": 100
        },
        "Service": {"Ref": "ECSDemoService"},
        "TaskDefinition": {"Ref": "BlueTaskDefinition"},
        "LoadBalancers": [
          {
            "ContainerName": "DemoApp",
            "ContainerPort": 80,
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"}
          }
        ]
      }
    },
    "PrimaryTaskSet": {
      "Type": "AWS::ECS::PrimaryTaskSet",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "Service": {"Ref": "ECSDemoService"},
        "TaskSetId": {"Fn::GetAtt": ["BlueTaskSet", "Id"]}
      }
    }
  }
}
```

Migration steps

Remove CodeDeploy-specific resources

You no longer need the following properties:

- The `AWS::CodeDeployBlueGreen` transform
- The `CodeDeployBlueGreenHook` hook
- The `GreenTaskDefinition` and `GreenTaskSet` resources (these will be managed by Amazon ECS)
- The `PrimaryTaskSet` resource (Amazon ECS will manage task sets internally)

Reconfigure the load balancer listener

Modify the `ALBListenerProdTraffic` resource to use a forward action with two target groups:

```
{
  "DefaultActions": [
    {
      "Type": "forward",
      "ForwardConfig": {
        "TargetGroups": [
          {
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
            "Weight": 1
          },
          {
            "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
            "Weight": 0
          }
        ]
      }
    }
  ]
}
```

Update the deployment properties

Update and add the following:

- Change the `DeploymentController` property from `EXTERNAL` to `ECS`.
- Add the `Strategy` property and set it to BLUE_GREEN.
- Add the `BakeTimeInMinutes` property.

```
{
  "DeploymentConfiguration": {
    "MaximumPercent": 200,
    "MinimumHealthyPercent": 100,
    "DeploymentCircuitBreaker": {
      "Enable": true,
      "Rollback": true
    },
    "BakeTimeInMinutes": 5,
    "Strategy": "BLUE_GREEN"
```

```
    }
  }
```

- Add the load balancer configuration to the service:

```
{
  "LoadBalancers": [
    {
      "ContainerName": "DemoApp",
      "ContainerPort": 80,
      "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
      "AdvancedConfiguration": {
        "AlternateTargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
        "ProductionListenerRule": {"Ref": "ALBListenerProdRule"},
        "RoleArn": {"Fn::GetAtt": ["ECSInfrastructureRoleForLoadBalancers", "Arn"]}
      }
    }
  ]
}
```

- Add the task definition reference to the service:

```
{
  "TaskDefinition": {"Ref": "BlueTaskDefinition"}
}
```

Create the AmazonECSInfrastructureRolePolicyForLoadBalancers role

Add a new IAM role that allows Amazon ECS to manage load balancer resources:

Replace the *`user-input`* with your value.

```
{
  "ECSInfrastructureRoleForLoadBalancers": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Sid": "AllowAccessToECSForInfrastructureManagement",
            "Effect": "Allow",
```

```
            "Principal": {
              "Service": "ecs.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "ManagedPolicyArns": [
        "arn:aws:iam::aws:policy/AmazonECSInfrastructureRolePolicyForLoadBalancers"
      ]
    }
  }
}
```

Testing recommendations

1. Deploy the migrated template to a non-production environment.
2. Verify that the service deploys correctly with the initial configuration.
3. Test a deployment by updating the task definition and observing the blue/green deployment process.
4. Verify that traffic shifts correctly between the blue and green deployments.
5. Test rollback functionality by forcing a deployment failure.

Template after migration

Final template

This is the complete AWS CloudFormation template using an Amazon ECS blue/green deployment:

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "Vpc": {
      "Type": "AWS::EC2::VPC::Id"
    },
    "Subnet1": {
      "Type": "AWS::EC2::Subnet::Id"
    },
    "Subnet2": {
      "Type": "AWS::EC2::Subnet::Id"
    }
```

```
  },
  "Resources": {
    "ExampleSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Security group for ec2 access",
        "VpcId": {"Ref": "Vpc"},
        "SecurityGroupIngress": [
          {
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 8080,
            "ToPort": 8080,
            "CidrIp": "0.0.0.0/0"
          },
          {
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIp": "0.0.0.0/0"
          }
        ]
      }
    },
    "ALBTargetGroupBlue": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
```

```
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ALBTargetGroupGreen": {
      "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
      "Properties": {
        "HealthCheckIntervalSeconds": 5,
        "HealthCheckPath": "/",
        "HealthCheckPort": "80",
        "HealthCheckProtocol": "HTTP",
        "HealthCheckTimeoutSeconds": 2,
        "HealthyThresholdCount": 2,
        "Matcher": {
          "HttpCode": "200"
        },
        "Port": 80,
        "Protocol": "HTTP",
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "TargetType": "ip",
        "UnhealthyThresholdCount": 4,
        "VpcId": {"Ref": "Vpc"}
      }
    },
    "ExampleALB": {
      "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
      "Properties": {
        "Scheme": "internet-facing",
        "SecurityGroups": [
          {"Ref": "ExampleSecurityGroup"}
        ],
        "Subnets": [
          {"Ref": "Subnet1"},
```

```
          {"Ref": "Subnet2"}
        ],
        "Tags": [
          {
            "Key": "Group",
            "Value": "Example"
          }
        ],
        "Type": "application",
        "IpAddressType": "ipv4"
      }
    },
    "ALBListenerProdTraffic": {
      "Type": "AWS::ElasticLoadBalancingV2::Listener",
      "Properties": {
        "DefaultActions": [
          {
            "Type": "forward",
            "ForwardConfig": {
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                },
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
                  "Weight": 0
                }
              ]
            }
          }
        ],
        "LoadBalancerArn": {"Ref": "ExampleALB"},
        "Port": 80,
        "Protocol": "HTTP"
      }
    },
    "ALBListenerProdRule": {
      "Type": "AWS::ElasticLoadBalancingV2::ListenerRule",
      "Properties": {
        "Actions": [
          {
            "Type": "forward",
            "ForwardConfig": {
```

```
              "TargetGroups": [
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
                  "Weight": 1
                },
                {
                  "TargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
                  "Weight": 0
                }
              ]
            }
          }
        ],
        "Conditions": [
          {
            "Field": "http-header",
            "HttpHeaderConfig": {
              "HttpHeaderName": "User-Agent",
              "Values": [
                "Mozilla"
              ]
            }
          }
        ],
        "ListenerArn": {"Ref": "ALBListenerProdTraffic"},
        "Priority": 1
      }
    },
    "ECSTaskExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "",
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
```

```
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
        ]
      }
    },
    "ECSInfrastructureRoleForLoadBalancers": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AllowAccessToECSForInfrastructureManagement",
              "Effect": "Allow",
              "Principal": {
                "Service": "ecs.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        },
        "ManagedPolicyArns": [
          "arn:aws:iam::aws:policy/AmazonECSInfrastructureRolePolicyForLoadBalancers"
        ]
      }
    },
    "BlueTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": {"Fn::GetAtt": ["ECSTaskExecutionRole", "Arn"]},
        "ContainerDefinitions": [
          {
            "Name": "DemoApp",
            "Image": "nginxdemos/hello:latest",
            "Essential": true,
            "PortMappings": [
              {
                "HostPort": 80,
                "Protocol": "tcp",
                "ContainerPort": 80
              }
            ]
          }
        ],
```

```
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "NetworkMode": "awsvpc",
        "Cpu": "256",
        "Memory": "512",
        "Family": "ecs-demo"
      }
    },
    "ECSDemoCluster": {
      "Type": "AWS::ECS::Cluster",
      "Properties": {}
    },
    "ECSDemoService": {
      "Type": "AWS::ECS::Service",
      "Properties": {
        "Cluster": {"Ref": "ECSDemoCluster"},
        "DesiredCount": 1,
        "DeploymentController": {
          "Type": "ECS"
        },
        "DeploymentConfiguration": {
          "MaximumPercent": 200,
          "MinimumHealthyPercent": 100,
          "DeploymentCircuitBreaker": {
            "Enable": true,
            "Rollback": true
          },
          "BakeTimeInMinutes": 5,
          "Strategy": "BLUE_GREEN"
        },
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "AssignPublicIp": "ENABLED",
            "SecurityGroups": [
              {"Ref": "ExampleSecurityGroup"}
            ],
            "Subnets": [
              {"Ref": "Subnet1"},
              {"Ref": "Subnet2"}
            ]
          }
        },
        "LaunchType": "FARGATE",
```

```
        "PlatformVersion": "1.4.0",
        "TaskDefinition": {"Ref": "BlueTaskDefinition"},
        "LoadBalancers": [
          {
            "ContainerName": "DemoApp",
            "ContainerPort": 80,
            "TargetGroupArn": {"Ref": "ALBTargetGroupBlue"},
            "AdvancedConfiguration": {
              "AlternateTargetGroupArn": {"Ref": "ALBTargetGroupGreen"},
              "ProductionListenerRule": {"Ref": "ALBListenerProdRule"},
              "RoleArn": {"Fn::GetAtt": ["ECSInfrastructureRoleForLoadBalancers",
 "Arn"]}
            }
          }
        ]
      }
    }
  }
}
```

Migrating from a CodeDeploy blue/green to an Amazon ECS rolling update service deployment

You can migrate your service deployments from an CodeDeploy blue/green deployment to an Amazon ECS rolling update deployment. This moves you away from CodeDeploy dependency to using an integrated deployment.

The Amazon ECS service scheduler replaces the currently running tasks with new tasks. The number of tasks that Amazon ECS adds or removes from the service during a rolling update is controlled by the service deployment configuration.

Prerequisites

Perform the following operations before you start a blue/green deployment.

1. You no longer need the Amazon ECS CodeDeploy IAM role.
2. Turn off CodeDeploy automation. For more information, see Working with deployment groups in CodeDeploy in the *CodeDeploy User Guide*.
3. Verify that there are no ongoing service deployments for the service. For more information, see View service history using Amazon ECS service deployments.

For more information about updating a service's deployment controller, see Update Amazon ECS service parameters.

Procedure

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.

   The cluster details page displays.
3. From the **Services** tab, choose the service.

   The service details page displays.
4. In the banner, choose **Migrate**.

   The **Update deployment configuration** page displays.
5. Expand **Deployment options**, and then specify the following parameters.

   a. For **Deployment controller type**, choose **ECS**.

   b. For **Deployment strategy**, choose **Rolling update**.

   c. For **Min running tasks**, enter the lower limit on the number of tasks in the service that must remain in the `RUNNING` state during a deployment, as a percentage of the desired number of tasks (rounded up to the nearest integer). For more information, see Deployment configuration.

   d. For **Max running tasks**, enter the upper limit on the number of tasks in the service that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer).
6. Expand **Load Balancing**, and then configure the following:

   a. For **Role**, choose the role that you created in the prerequisites with the blue/green permissions.

      For more information, see Permissions required for Lambda functions in Amazon ECS blue/green deployments.

   b. For **Listener**, choose the production listener from your CodeDeploy blue/green deployment.

   c. For **Target group**, choose the production target group from your CodeDeploy blue/green deployment.

7. Choose **Update**.

Next steps

You must update the service for the change to take effect. For more information, see Updating an Amazon ECS service.

Update the Amazon ECS deployment strategy

Amazon ECS supports multiple deployment strategies for updating your services. You can migrate between these strategies based on your application requirements. This topic explains how to migrate between rolling deployments and blue/green deployments.

Understanding Amazon ECS deployment strategies

Before migrating between deployment strategies, it's important to understand how each strategy works and their key differences:

Rolling deployments

In a rolling deployment, Amazon ECS replaces the current running version of your application with a new version. The service scheduler uses the minimum and maximum healthy percentage parameters to determine the deployment strategy.

Rolling deployments are simpler to set up but provide less control over the deployment process and traffic routing.

Blue/green deployments

In a blue/green deployment, Amazon ECS creates a new version of your service (green) alongside the existing version (blue). This allows you to verify the new version before routing production traffic to it.

Blue/green deployments provide more control over the deployment process, including traffic shifting, testing, and rollback capabilities.

Best practices

Follow these best practices when migrating between deployment strategies:

- **Test in a non-production environment**: Always test the update in a non-production environment before applying changes to production services.

- **Plan for rollback**: Have a rollback plan in case the update doesn't work as expected.
- **Monitor during transition**: Closely monitor your service during and after the migration to ensure it continues to operate correctly.
- **Update documentation**: Update your deployment documentation to reflect the new deployment strategy.
- **Consider traffic impact**: Understand how the update might impact traffic to your service and plan accordingly.

Updating the deployment strategy from rolling update to Amazon ECS blue/green

You can migrate from a rolling update deployment to an Amazon ECS blue/green deployment when you want to make and test service changes before implementing them in a production environment.

Prerequisites

Before migrating your service from rolling to blue/green deployments, ensure you have the following:

- Wait for any current deployments to complete.
- An existing Amazon ECS service using the rolling deployment strategy.
- If you have multiple service revisions serving traffic, Amazon ECS attempts to consolidate traffic to a single revision during migration. If this fails, you might need to manually update your service to use a single revision before migrating.
- Configure the appropriate permissions.
  - For information about Elastic Load Balancing permissions, see [Amazon ECS infrastructure IAM role for load balancers](#).
  - For information about Lambda permissions, see [Permissions required for Lambda functions in Amazon ECS blue/green deployments](#).
- Depending on configuration, you need to perform one of the following:
  - If your service uses Elastic Load Balancing, update your service with the new `advancedConfiguration` and start a rolling deployment.
  - If your service uses Service Connect, update your service and start a rolling deployment.
  - If your service uses both Elastic Load Balancing and Service Connect, perform both steps above (you can use a single UpdateService request).

- If your service uses none of the above, then no additional operation is needed.
- Amazon ECS blue/green deployments require that your service uses one of the following features. Configure the appropriate resources.
  - Application Load Balancer - For more information, see Application Load Balancer resources for blue/green deployments.
  - Network Load Balancer - For more information, see Network Load Balancer resources for Amazon ECS blue/green deployments.
  - Service Connect - For more information, see Service Connect resources for Amazon ECS blue/green deployments.

Procedure

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster that contains the service you want to migrate.

   The Cluster details page is displayed.
4. On the **Cluster details** page, choose the **Services** tab.
5. Choose the service, and then choose **Update**.

   The Update service page is displayed
6. Expand **Deployment options**, and then do the following:
7. For **Deployment strategy**, choose **Blue/green**.
8. Configure the blue/green deployment settings:

   a. For **Bake time**, enter the number of minutes that both the blue and green service revisions will run simultaneously before the blue revision is terminated.

      This allows time for verification and testing.

   b. (Optional) Configure Lambda functions to run at specific stages of the deployment. Under **Deployment lifecycle hooks**, configure Lambda functions for the following stages:

      - **Pre scale up**: Runs before scaling up the green service revision
      - **Post scale up**: Runs after scaling up the green service revision
      - **Test traffic shift**: Runs during test traffic routing to the green service revision

- **Post test traffic shift**: Runs after test traffic is routed to the green service revision
- **Production traffic shift**: Runs during production traffic routing to the green service revision
- **Post production traffic shift**: Runs after production traffic is routed to the green service revision

To add a lifecycle hook:

a. Choose **Add**.

b. For **Lambda function**, enter the function name or ARN.

c. For **Role**, choose the IAM role that has permission to invoke the Lambda function.

d. For **Lifecycle stages**, select the stages when the Lambda function should run.

e. Optional: For **Hook details**, enter key-value pairs to provide additional information to the hook.

9. Configure the load balancer settings:

   a. Under **Load balancing**, verify that your service is configured to use a load balancer.

   b. For **Target group**, choose the primary target group for your production (blue) environment.

   c. For **Alternate target group**, choose the target group for your test (green) environment.

   d. For **Production listener rule**, choose the listener rule for routing production traffic.

   e. Optional: For **Test listener rule**, choose a listener rule for routing test traffic to your green environment.

   f. For **Role**, choose the IAM role that allows Amazon ECS to manage your load balancer.

10. Review your configuration changes, and then choose **Update**.

Next steps

- Update the service to start the deployment. For more information, see Updating an Amazon ECS service.
- Monitor the deployment process to ensure it follows the blue/green pattern:
  - The green service revision is created and scaled up
  - Test traffic is routed to the green revision (if configured)

- Production traffic is shifted to the green revision
- After the bake time, the blue revision is terminated

Updating the deployment strategy from Amazon ECS blue/green to rolling update

You can migrate a blue/green deployment to a rolling update deployment.

Keep the following considerations in mind when migrating to rolling deployments:

- **Traffic handling**: With rolling deployments, new tasks start receiving traffic as soon as they pass health checks. There is no separate testing phase as with blue/green deployments.
- **Resource efficiency**: Rolling deployments typically use fewer resources than blue/green deployments because they replace tasks incrementally rather than creating a complete duplicate environment.
- **Rollback complexity**: Rolling deployments make rollbacks more complex compared to blue/green deployments. If you need to roll back, you must initiate a new deployment with the previous task definition.
- **Deployment speed**: Rolling deployments may take longer to complete than blue/green deployments, especially for services with many tasks.
- **Load balancer configuration**: Your existing load balancer configuration will continue to work with rolling deployments, but the traffic shifting behavior will be different.

Prerequisites

Before migrating your service from blue/green to rolling deployments, ensure you have the following:

- An existing Amazon ECS service using the blue/green deployment strategy
- No ongoing deployments for the service (wait for any current deployments to complete)
- A clear understanding of how your service will behave with rolling deployments

> **Note**
>
> You cannot migrate a service to rolling deployment if it has an ongoing deployment. Wait for any current deployments to complete before proceeding.

Migration procedure

Follow these steps to migrate your Amazon ECS service from blue/green to rolling deployments:

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster that contains the service you want to migrate.
4. On the **Cluster details** page, choose the **Services** tab.
5. Select the service you want to migrate, and then choose **Update**.
6. On the **Update service** page, navigate to the **Deployment options** section and expand it if necessary.
7. For **Deployment strategy**, choose **Rolling update**.
8. Configure the rolling deployment settings:

    a. For **Minimum healthy percent**, enter the minimum percentage of tasks that must remain in the `RUNNING` state during a deployment. This value is specified as a percentage of the desired number of tasks for the service.

    b. For **Maximum percent**, enter the maximum percentage of tasks that are allowed in the `RUNNING` or `PENDING` state during a deployment. This value is specified as a percentage of the desired number of tasks for the service.

9. Optional: Under **Deployment failure detection**, configure how Amazon ECS detects and handles deployment failures:

    a. To enable the deployment circuit breaker, choose **Use the deployment circuit breaker**.

    b. To automatically roll back failed deployments, choose **Rollback on failure**.

10. Review your configuration changes, and then choose **Update** to save your changes and migrate the service to rolling deployment.

Amazon ECS will update your service configuration to use the rolling deployment strategy. The next time you update your service, it will use the rolling deployment process.

> **Note**
>
> When you migrate from blue/green to rolling deployment, Amazon ECS handles the transition by:

1. Identifying the current active service revision that is serving traffic.
2. Maintaining the existing load balancer configuration but changing how new deployments are handled.
3. Preparing the service for future rolling deployments.

Next steps

- Update the service to start the deployment. For more information, see [Updating an Amazon ECS service](#).

Troubleshooting Amazon ECS deployment strategy updates

This section provides solutions for common issues you might encounter when migrating deployment strategies.

Multiple service revisions or tasks sets

The following issues relate to having multiple service revisions for a deployment.

Multiple task sets when updating ECS deployment controller

Error message: `Updating the deployment controller is not supported when there are multiple tasksets in the service. Please ensure your service has only one taskset and try again.`

Solution: This error occurs when attempting to change the deployment controller type of a service with multiple active task sets. To resolve this issue for the `CODE_DEPLOY` or `EXTERNAL` deployment controller:

1. Check current task sets:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].taskSets"
```

2. Wait for any in-progress deployments to complete.
3. Force a new deployment to clean up task sets:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --force-new-deployment
```

4. If necessary, delete extra task sets manually:

```
aws ecs delete-task-set --cluster your-cluster-name --service your-service-name
 --task-set task-set-id
```

5. After only one task set remains, retry updating the deployment controller.

For more information, see the section called "Service deployment controllers and strategies".

Missing primary task set when updating ECS deployment controller

Error message: `Updating the deployment controller requires a primary taskset in the service. Please ensure your service has a primary taskset and try again.`

Solution: This error occurs when attempting to change the deployment controller type of a service that doesn't have a primary task set. To resolve this issue:

1. Verify the service status and task sets.). If a task set exists in the service, it should be marked as `ACTIVE`.

   ```
   aws ecs describe-services --cluster your-cluster-name --services your-service-
   name --query "services[0].taskSets[*].[status,id]
   ```

   If there are no task sets in the `ACTIVE` state, migrate the deployment. For more information, see Migration approaches.

2. If the service has no running tasks, deploy at least one task by updating the service:

   ```
   aws ecs update-service-primary-task-set --cluster your-cluster-name --
   service your-service-name --primary-task-set your-taskset-id
   ```

   This will mark the (previously `ACTIVE`) task set in the service as `PRIMARY` status.

3. Wait for the task to reach a stable running state. You can check the status with:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deployments"
```

4. After the service has a primary task set with running tasks, retry updating the deployment controller.

For more information, see [the section called “Service deployment controllers and strategies”](#).

Mismatch between the deployment failure detection type and deployment controller

The following issues relate to a mismatch between the deployment failure detection type and deployment controller.

Deployment circuit breaker with non-ECS controller

Error message: `Deployment circuit breaker feature is only supported with ECS deployment controller. Update to ECS deployment controller and try again.`

Solution: This error occurs when attempting to enable the deployment circuit breaker feature on a service that is not using the ECS deployment controller. The deployment circuit breaker is only compatible with the ECSdeployment controller.

1. Check your service's current deployment controller:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].deploymentController"
```

2. Update your service to use the ECS deployment controller:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. After the service is using the ECS deployment controller, enable the deployment circuit breaker:

```
aws ecs update-service --cluster your-cluster-name --
service your-service-name --deployment-configuration
 "deploymentCircuitBreaker={enable=true,rollback=true}"
```

For more information, see the section called " How the deployment circuit breaker detects failures".

Alarm-based rollback with non-ECS controller

Error message: `Alarm based rollback feature is only supported with ECS deployment controller. Update to ECS deployment controller and try again.`

Solution: This error occurs when attempting to configure alarm-based rollback on a service that is not using the ECS deployment controller. The alarm-based rollback feature is only compatible with the ECS deployment controller.

1. Check your service's current deployment controller:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].deploymentController"
```

2. Update your service to use the ECS deployment controller:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. After the service is using the ECS deployment controller, configure alarm-based rollback:

```
aws ecs update-service --cluster your-cluster-name --services your-service-name --deployment-configuration "alarms={alarmNames=[your-alarm-name],enable=true,rollback=true}"
```

For more information, see the section called "How CloudWatch alarms detect deployment failures".

Mismatch between Service Connect and the deployment controller

The following issues relate to a mismatch between Service Connect and the deployment controller.

EXTERNAL controller with Service Connect

Error message: `The EXTERNAL deployment controller type is not supported for services using Service Connect.`

Solution: This error occurs when attempting to use the EXTERNAL deployment controller with an service that has Service Connect enabled. The EXTERNAL controller is not compatible with Service Connect.

1. Check if your service has Service Connect enabled:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].serviceConnectConfiguration"
```

2. If you need to use the EXTERNAL deployment controller, disable Service Connect by updating your service:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-connect-configuration "{}"
```

3. Alternatively, if you must use Service Connect, use the ECS deployment controller instead:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

For more information, see the section called "Service deployment controllers and strategies".

Service Connect with non-ECS controller

Error message: `Service Connect feature is only supported with ECS (rolling update) deployment controller. Update to ECS deployment controller and try again.`

Solution: This error occurs when attempting to enable Service Connect on a that is not using the ECS deployment controller. The Service Connect feature is only compatible with the ECS deployment controller.

1. Check your service's current deployment controller:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].deploymentController"
```

2. Update your service to use the ECS deployment controller:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

3. Once the service is using the ECS deployment controller, enable Service Connect:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-connect-configuration "enabled=true,namespace=your-namespace"
```

For more information, see the section called "Service deployment controllers and strategies".

Mismatch between controller type and scheduling strategy

The following issues relate to a mismatch between controller type and scheduling strategy.

`CODE_DEPLOY` controller with `DAEMON` scheduling strategy

Error message: `The CODE_DEPLOY deployment controller type is not supported for services using the DAEMON scheduling strategy.`

Solution: This error occurs when attempting to use the CODE_DEPLOY deployment controller with a service that uses the `DAEMON` scheduling strategy. The `CODE_DEPLOY` controller is only compatible with the `REPLICA` scheduling strategy.

1. Check your service's current scheduling strategy:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].schedulingStrategy"
```

2. If you need blue/green deployments, change your service to use the `REPLICA` scheduling strategy:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --scheduling-strategy REPLICA
```

3. Alternatively, if you must use the `DAEMON` scheduling strategy, use the `ECS` deployment controller instead:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

For more information, see the section called "Service deployment controllers and strategies".

EXTERNAL controller with DAEMON scheduling strategy

Error message: `The EXTERNAL deployment controller type is not supported for services using the DAEMON scheduling strategy.`

Solution: This error occurs when attempting to use the EXTERNAL deployment controller with an ECS service that uses the DAEMON scheduling strategy. The EXTERNAL controller is only compatible with the REPLICA scheduling strategy.

1. Check your service's current scheduling strategy:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-
name --query "services[0].schedulingStrategy"
```

2. If you need to use the `EXTERNAL` deployment controller, change your service to use the `REPLICA` scheduling strategy:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --scheduling-strategy REPLICA
```

3. Alternatively, if you must use the `DAEMON` scheduling strategy, use the `ECS` deployment controller instead:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --deployment-controller type=ECS
```

For more information, see the section called "Service deployment controllers and strategies".

Service registries with external launch type

Error message: `Service registries are not supported for external launch type.`

Solution: This error occurs when attempting to configure service discovery (service registries) for a service that uses the EXTERNAL launch type. Service discovery is not compatible with the EXTERNAL launch type.

1. Check your service's current launch type:

```
aws ecs describe-services --cluster your-cluster-name --services your-service-name --query "services[0].launchType"
```

2. If you need service discovery, change your service to use either the EC2 or FARGATE launch type:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --launch-type FARGATE
```

3. Alternatively, if you must use the EXTERNAL launch type, remove the service registry configuration:

```
aws ecs update-service --cluster your-cluster-name --service your-service-name
 --service-registries "[]"
```

For more information, see the section called “Service deployment controllers and strategies”.

Revert a deployment controller update

If you decide that you want to return to the previous deployment controller, you can do one of the following:

- If you used AWS CloudFormation, you can use the previous template to create a new stack. For more information, see Create a stack from in the *AWS CloudFormation User Guide*.
- If you used the Amazon ECS console, or the AWS CLI, you can update the service. For more information, see the section called “Updating a service”.

  If you use the update-service command, use the `--deployment-controller` option and set it to the previous deployment controller.

View service history using Amazon ECS service deployments

Service deployments provide a comprehensive view of your deployments. Service deployments provide the following information about the service:

- The currently deployed workload configuration (the source service revision)
- The workload configuration being deployed (the target service revision)
- The deployment status
- The number of failed tasks the circuit break detected
- The CloudWatch alarms that are in alarm
- When the service deployment started and completed
- The details of a rollback if one occurred

For information about the service deployment properties, see Properties included in an Amazon ECS service deployment.

Service deployments are read-only and each have a unique ID.

There are three service deployment stages:

Stage	Definition	Associated states
Pending	A service deployment has been created, but has not started	PENDING
Ongoing	A service deployment is in-progress	• IN_PROGRESS • STOP_REQUESTED • ROLLBACK_REQUESTED • ROLLBACK_IN_PROGRESS
Completed	A service deployment has finished (successfully or unsuccessfully)	• SUCCESSFUL • STOPPED • ROLLBACK_SUCCESSFUL • ROLLBACK_FAILED

You use service deployments to understand the lifecycle of your service and to determine if there are any actions you need to take. For example, if a rollback happened, you might need to investigate the service deployment and looking at service events.

You can view the most recent 90-day history for deployments created on or after October 25, 2024 by using the console, API, and the AWS CLI.

You can stop a deployment that has not completed. For more information, see Stopping Amazon ECS service deployments.

Service deployment lifecycle

Amazon ECS creates a new service deployment automatically when any of the following actions happen:

- A user creates a service.
- A user updates the service and uses the force new deployment option.
- A user updates one or more service properties that require a deployment.

While a deployment is ongoing, Amazon ECS updates the following service deployment properties to reflect the service deployment's progress:

- The state
- The number of running tasks

  The number of running tasks indicated in the service revision might not equal the actual number of running task. This number represents the number of tasks running when the deployment completed. For example, if you launched tasks independent of the service deployment, those tasks are not included in the running task count for the service revision.
- Circuit breaker failure detection:
  - The number of tasks that have failed to start
- CloudWatch alarm failure detection
  - The alarms that are active
- Rollback information:
  - The start time
  - The reason for the rollback
  - The ARN of the service revision used for the rollback

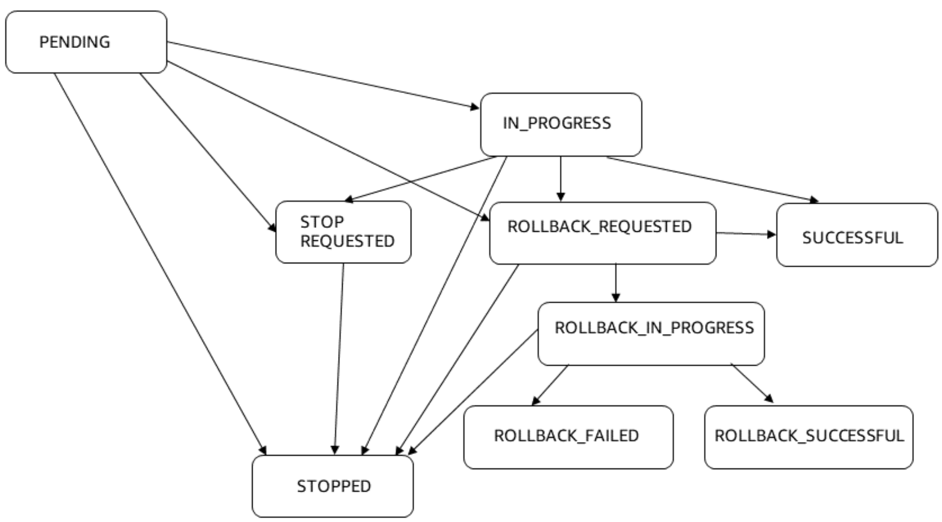
- The status reason

Amazon ECS deletes the service deployment when you delete a service.

Service deployment states

A service deployment starts in `PENDING` state.

The following illustration shows the service deployment states that can happen after the `PENDING` state: `IN_PROGRESS`, `ROLLBACK_REQUESTED`, `SUCCESSFUL`, `STOP_REQUESTED`, `ROLLBACK_IN_PROGRESSS`, `ROLLBACK_FAILED`, `ROLLBACK_SUCCESSFUL`, and `STOPPED`.



The following information provides details about service deployment states:

- `PENDING` - The service deployment has been created, but has not started.

  The state can move to `IN_PROGRESS`, `ROLLBACK_REQUESTED`, `STOP_REQUESTED`, or `STOPPED`.
- `IN_PROGRESS` - The service deployment is ongoing.

  The state can move to `SUCCESSFUL`, `STOP_REQUESTED`, `ROLLBACK_REQUESTED`, `ROLLBACK_IN_PROGRESS`, and `STOPPED`.

- `STOP_REQUESTED` - The service deployment state moves to `STOP_REQUESTED` when any of the following happen:
  - A user starts a new service deployment.
  - The rollback option is not in use for the failure detection mechanism (the circuit breaker or alarm-based) and the service does not reach the `SUCCESSFUL` state.

  The state moves to `STOPPED`.
- `ROLLBACK_REQUESTED` - The service deployment state moves to `ROLLBACK_REQUESTED` when a user request a rollback through the console, API, or CLI.

  The state can move to `SUCCESSFUL`, `ROLLBACK_IN_PROGRESS`, and `STOPPED`.
- `SUCCESSFUL` - The service deployment state moves to `SUCCESSFUL` when the service deployment successfully completes.
- `ROLLBACK_IN_PROGRESS` - The service deployment state moves to `ROLLBACK_IN_PROGRESS` when the rollback option is in use for the failure detection mechanism (the circuit breaker or alarm-based) and the service fails.

  The state moves to `ROLLBACK_SUCCESSFUL`, or `ROLLBACK_FAILED`.

Properties included in an Amazon ECS service deployment

The following properties are included in a service deployment.

Property	Description	
Service deployment ARN	The ARN of the service deployment.	
Service ARN	The ARN of the service for this service deployment.	
Cluster ARN	The ARN for the cluster that hosts the service.	
Service deployment creation time	The time the service deployment was created.	

Property	Description
Service deployment start time	The time the service deployment started.
Service deployment finish time	The time the service deployment finished.
Service deployment stopped time	The time the service deployment stopped.
Service deployment update time	The time that the service deployment was last updated.
Source service revisions	The currently running service revisions. For information about the included properties, see Properties included in an Amazon ECS service revision.

Property	Description
Deployment configuration	The deployment parameters including the circuit breaker configuration, the alarms that determine. • The lower limit on the number of tasks that should be running for a service during a deploymen t or when a container instance is draining, as a percent of the desired number of tasks for the service. • The upper limit on the number of tasks that should be running for a service during a deploymen t or when a container instance is draining, as a percent of the desired number of tasks for the service. • The circuit breaker configuration. • The alarms used for failure detection.
Target service revision	The service revision to deploy. After the deployment completes successfully, the target service revision is the running service revision.

Property	Description
Service deployment status	The service deployment state. The valid values are PENDING, SUCCESSFUL, STOPPED, STOP_REQU ESTED, STOP_IN_P ROGRESS, IN_PROGRESS, ROLLBACK_IN_PROGRESS, ROLLBACK_SUCCESSFUL, and ROLLBACK_FAILED.
Service deployment status information	Information about why the service deployment is in the current status. For example, the circuit breaker detected a failure.
Rollback information	The rollback options the service deployment uses when the deployment fails.
Service deployment circuit breaker options	The circuit breaker that determines a service deployment failed.
CloudWatch alarms for the service deployment	The CloudWatch alarms that determine when a service deployment fails.

Permissions required for viewing Amazon ECS service deployments

When you follow the best practice of granting least privilege, you need to add additional permissions in order to view service deployments in the console.

You need access to the following actions:

- ListServiceDeployments
- DescribeServiceDeployments
- DescribeServiceRevisions

You need access to the following resources:

- Service
- Service deployment
- Service revision

The following example policy contains the required permissions, and limits the actions to a specified service.

Replace the `account`, `cluster-name`, and `service-name` with your values.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "ecs:ListServiceDeployments",
              "ecs:DescribeServiceDeployments",
              "ecs:DescribeServiceRevisions"
            ],
            "Resource": [
             "arn:aws:ecs:us-east-1:123456789012:service/cluster-name/service-name",
             "arn:aws:ecs:us-east-1:123456789012:service-deployment/cluster-name/
service-name/*",
             "arn:aws:ecs:us-east-1:123456789012:service-revision/cluster-name/service-
name/*"
            ]
        }
    ]
}
```

Viewing Amazon ECS service deployments

You can see the most recent 90-day history for deployments created on or after October 25, 2024. The service deployments can be in any of the following states:

- In-progress
- Pending
- Completed

You can use this information to determine if you need to update how the service is being deployed, or the service revision. For information about the included properties, see Properties included in an Amazon ECS service deployment.

Before you begin, configure the required permissions for viewing service deployments. For more information, see Permissions required for viewing Amazon ECS service deployments.

Amazon ECS Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, choose the service.

   The service details page displays.
4. On the service details page, choose **Deployments**.
5. Choose the service deployment to view.

To view service deployments for this deployment type	Do this
Ongoing deployment	Under **Ongoing deployments**, choose the **Deployment ID**.
Last deployment	Under **Last deployment**, choose the **Deployment ID**.
Completed deployments	Under **Service deployments**, choose the **Deployment ID**.

The service deployment details page appears.

6. (Optional) Compare service revisions to view the differences.

   Under **Service revisions**, choose **Compare revisions**, and then select 2 revisions to compare.

   The service revisions are displayed side-by-side with the differences highlighted.

AWS CLI

1. Run `list-service-deployments` to retrieve the service deployment ARN.

   Replace the variables with your values.

   ```
   aws ecs list-service-deployments --cluster cluster-name --service service-name
   ```

   Note the serviceDeploymentArn for the deployment you want to view.

   ```
   {
       "serviceDeployments": [
           {
               "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-deployment/example/sd-example/NCWGC2ZR-taawPAYrIaU5",
               "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/example/sd-example",
               "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/example",
               "targetServiceRevisionArn": "arn:aws:ecs:us-west-2:123456789012:service-revision/example/sd-example/4980306466373577095",
               "status": "SUCCESSFUL"
           }
       ]
   }
   ```

2. Run `describe-service-deployments`. Use the `serviceDeploymentArn` that was returned from `list-service-deployments`.

   Replace the variables with your values.

```
aws ecs describe-service-deployments --service-deployment-arns
 arn:aws:ecs:region:123456789012:service-deployment/cluster-name/service-
name/NCWGC2ZR-taawPAYrIaU5
```

Next steps

You can view the details for service revisions in the deployment. For more information, see [Viewing Amazon ECS service revision details](#)

Amazon ECS service revisions

A service revision contains a record of the workload configuration Amazon ECS is attempting to deploy. Whenever you create or deploy a service, Amazon ECS automatically creates and captures the configuration that you're trying to deploy in the service revision.

Service revisions are read-only and have unique identifiers. For information about the included properties, see Properties included in an Amazon ECS service revision.

Service revisions provide the following benefits:

- During a service deployment, you can compare the currently deployed service revision (source) with the one being deployed (target).
- When you use the rollback option for a service deployment, Amazon ECS automatically rolls back the service deployment to the last successfully deployed service revision.
- Service revisions contain the record of the workload configuration in one resource.

Service revision lifecycle

Amazon ECS automatically creates a new service revision when you create a service, or update a service property that starts a deployment.

Amazon ECS doesn't create a new service revision for a rollback operation. Amazon ECS uses the last successful service revision for the rollback.

A service revision is immutable.

Amazon ECS deletes the service revision when you delete a service.

You can view service revisions created on or after October 25, 2024 by using the console, API, and the CLI.

Properties included in an Amazon ECS service revision

The following properties are included in a service revision.

Resource	Description	
Service ARN	The ARN that identifies the service.	
Cluster ARN	The ARN for the cluster that hosts the service.	
Task definition ARN	The ARN of the task definition used for the service tasks.	
Service registries	The details for the service registries used for service discovery. • The container name The container name value to be used for your service discovery service. • The container port The port value to be used for your service discovery service. This value is also specified in the task definition. • The port The port value used if your service discovery service specified an SRV record.	

Resource	Description
	• The registry ARN The Amazon Resource Name (ARN) of the service registry.
Capacity providers	The capacity provider strategy details. • The capacity provider name The capacity provider name used for the service. • The capacity provider weight The relative percentag e of the total number of tasks launched that should use the specified capacity provider. • The capacity provider base The number of tasks, at a minimum, to run on the specified capacity provider.
Container images	The details about the container images. • The container name • The image digest • The container image

Resource	Description
Networking	The network configuration for the service. • Whether there is a public IP address • The subnets the tasks run in • The security groups that control the service traffic
Launch type	The launch type used for the service.

Fargate-specific properties	When the launch type is Fargate, this is information about the Fargate version.
	• The platform version and the platform family
	• Ephemeral storage
	  ◦ The amount of ephemeral storage to allocate for the deployment.
	  ◦ The KMS key to use for storage encryption.

Resource	Description	
Amazon EBS volumes that are configured at deployment	The configuration for a volume specified in the task definition as a volume that is configured at launch time. • Encrypted Indicates whether the volume is encrypted. • File system type The Linux filesystem type for the volume. • IOPS The number of I/O operations per second (IOPS). • KMS key • IAM role The ARN of the IAM role to associate with this volume. • Size The size of the volume in GiB. • Snapshot ID The volume snapshot ID. • Tag specifications The tag configuration for the volume. • Throughput	

Resource	Description
	The throughput provisioned for the volume. • Volume type The volume type.
Service Connect	The Service Connect configuration. • The indication of whether Service Connect is in use • The namespace • The list of interconnected services • The Service Connect logging configuration
Service load balancers	The load balancers that route the service traffic. • The name • The container name • The container port • The target group ARN
Runtime Monitoring	Indicates if Runtime Monitoring is on.
Creation date	The date the service revision was created.
VPC Lattice	The VPC Lattice configuration for the service revision.

Viewing Amazon ECS service revision details

You can view information about the following service revision types that were created on or after October 25, 2024:

- Source -The currently deployed workload configuration
- Target - The workload configuration being deployed

Amazon ECS Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, choose the service.

   The service details page displays.
4. On the service details page, choose **Deployments**.
5. Choose the service revision to view.

To view service revisions for this deployment type	Do this	
Ongoing deployments	Under **Ongoing deployments**, do the following: • To view the source service revision, under **Service revisions**, choose the service revision ID for the Source **Revision type**. • To view the target service revision, under **Service revisions**, choose the service revision ID for the Target **Revision type**.	

To view service revisions for this deployment type	Do this
Last deployment	Under **Last deployment**, choose the **Target service revision**.
Completed deployments	Under **Service deployments**, do the following: • Under **Target service revision**, choose the ID.

AWS CLI

1. Run `describe-service-deployments` to retrieve the service revision ARN.

   Replace the variables with your values.

```
aws ecs describe-service-deployments --service-deployment-arns
 arn:aws:ecs:region:account-id:service/cluster-name/service-name/NCWGC2ZR-
taawPAYrIaU5
```

   Note the `arn` for the `sourceServiceRevisions` or the `targetServiceRevisions`.

```
{
    "serviceDeployments": [
        {
            "serviceDeploymentArn": "arn:aws:ecs:us-west-2:123456789012:service-
deployment/example/sd-example/NCWGC2ZR-taawPAYrIaU5",
            "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/example/
sd-example",
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/example",
            "updatedAt": "2024-09-10T16:49:35.572000+00:00",
            "sourceServiceRevision": {
                "arn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
example/sd-example/4980306466373578954",
                "requestedTaskCount": 0,
                "runningTaskCount": 0,
                "pendingTaskCount": 0
```

```
            },
            "targetServiceRevision": {
                "arn": "arn:aws:ecs:us-west-2:123456789012:service-revision/
example/sd-example/4980306466373577095",
                "requestedTaskCount": 0,
                "runningTaskCount": 0,
                "pendingTaskCount": 0
            },
            "status": "IN_PROGRESS",
            "deploymentConfiguration": {
                "deploymentCircuitBreaker": {
                    "enable": false,
                    "rollback": false
                },
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            }
        }
    ],
    "failures": []
}
```

2. Run `describe-service-revisions`. Use the `arn` that was returned from `describe-service-deployments`.

   Replace the variables with your values.

   ```
   aws ecs describe-service-revisions --service-revision-arns
    arn:aws:ecs:region:123456789012:service-revision/cluster-name/service-
   name/4980306466373577095
   ```

Balancing an Amazon ECS service across Availability Zones

To help your applications achieve high availability, we recommend configuring your multi-task services to run across multiple Availability Zones. For services that specify their first placement strategy to be Availability Zone spread, AWS makes a best effort to evenly distribute service tasks across the available Availability Zones.

However, there might be times when the number of tasks running in one Availability Zone is not the same as in other Availability Zones, such as after an Availability Zone disruption. To address this task imbalance, you can enable the Availability Zone rebalancing feature.

With Availability Zone rebalancing, Amazon ECS continuously monitors the distribution of tasks across Availability Zones for each of your services. When Amazon ECS detects an uneven task distribution, it automatically takes action to rebalance the workload across Availability Zones. This involves launching new tasks in the Availability Zones with the fewest tasks and terminating tasks in the overloaded Availability Zones.

This redistribution ensures no single Availability Zone becomes a point of failure, helping maintain the overall availability of your containerized applications. The automated rebalancing process eliminates the need for manual intervention, speeding the time to recovery after an event.

The following is an overview of the Availability Zone rebalancing process:

1. Amazon ECS starts monitoring a service after it reaches the steady state, and looks at the number of tasks running in each Availability Zone.
2. Amazon ECS performs the following operations when it detects an imbalance in the number of tasks running in each Availability Zone:
   - Sends a service event indicating that Availability Zone rebalancing is starting.
   - Starts tasks in Availability Zones with the fewest number of running tasks
   - Stops the tasks in Availability Zones with the largest number of running tasks.
   - The scheduler waits for the newly started tasks to be `HEALTHY` and `RUNNING` before stopping the tasks in the over-scaled Availability Zone.
   - Sends a service event with the Availability Zone rebalancing outcome.

How Amazon ECS detects uneven task distribution

Amazon ECS determines an imbalance in the number of tasks running in each Availability Zone by dividing the service's desired task count by the number of configured Availability Zones. If the desired task count doesn't divide evenly, Amazon ECS distributes the remainder of tasks evenly across the configured Availability Zones. Each Availability Zone must have at least one task.

For example, consider an Amazon ECS service with a desired count of two tasks configured for two Availability Zones. In this scenario, the desired task count divides evenly. A balanced distribution would be one task per Availability Zone. If there are two tasks in Availability Zone 1 and zero tasks in Availability Zone 2, Amazon ECS would initiate rebalancing by starting a task in Availability Zone 2 before stopping a task in Availability Zone 1.

Now, consider an Amazon ECS service with a desired count of three tasks configured for two Availability Zones. In this scenario, the desired task count does not divide evenly. A balanced distribution would be one task in Availability Zone 1 and two tasks in Availability Zone 2 because each Availability Zone has at least one task and the remainder task is placed in Availability Zone 2.

Consider an Amazon ECS service that has a desired count of five tasks configured for three Availability Zones. In this scenario, the desired task count does not divide evenly. A balanced distribution would be one task in Availability Zone 1 and two tasks each in Availability Zones 2 and 3. After accounting for every Availability Zone having one task each, the two remainder tasks are distributed evenly across the Availability Zones.

Considerations for configuring Availability Zone rebalancing

Consider the following when you want to configure Availability Zone rebalancing:

- Availability Zone rebalancing supports the Fargate and EC2 launch types. For Fargate, Amazon ECS will automatically redistribute tasks across available Availability Zones to maintain balance. For the EC2 launch type, Amazon ECS rebalances tasks across existing container instances on a best-effort basis, respecting your defined placement strategies and constraints. However, Amazon ECS can't launch new instances in underutilized Availability Zones as part of the rebalancing process, limiting the rebalancing to existing container instances.
- Availability Zone rebalancing works in the following configurations:
  - Services that use the `Replica` strategy
  - Services that specify Availability Zone spread as the first task placement strategy, or do not specify a placement strategy.
- You can't use Availability Zone rebalancing with services that meet any of the following criteria:
  - Uses the `Daemon` strategy
  - Uses the `EXTERNAL` launch type (ECS Anywhere)
  - Uses 100% for the `maximumPercent` value
  - Uses a Classic Load Balancer
  - Uses the `attribute:ecs.availability-zone` as a task placement constraint

Placement strategies and placement constraints with Availability Zone rebalancing

Placement strategies determine how Amazon ECS selects container instances and Availability Zones for task placement termination. Task placement constraints are rules that determine whether a task is allowed to run on a specific container instance.

For the EC2 launch type, you can use placement strategies and placement constraints in conjunction with Availability Zone rebalancing. However, for Availability Zone rebalancing to work, the Availability Zone spread placement strategy must be the first strategy specified.

Availability Zone rebalancing is compatible with various placement strategy combinations. For example, you can create a strategy that first distributes tasks evenly across Availability Zones, and then bin packs tasks based on memory within each Availability Zone. In this case, Availability Zone rebalancing works because the Availability Zone spread strategy is specified first.

It's important to note that Availability Zone rebalancing won't work if the first strategy in the placement strategy array is not an Availability Zone spread component. This requirement ensures that the primary focus of task distribution is maintaining balance across Availability Zones, which is crucial for high availability.

For more information about task placement strategies and constraints, see How Amazon ECS places tasks on container instances.

Task placement strategies and constraints aren't supported for tasks using the Fargate launch type. Fargate will try its best to spread tasks across accessible Availability Zones. If the capacity provider includes both Fargate and Fargate Spot, the spread behavior is independent for each capacity provider.

The following example strategy distributes tasks evenly across Availability Zones, and then bin packs tasks based on memory within each Availability Zone. Availability Zone rebalancing is compatible with the service because the `spread` strategy is first.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "memory",
```

```
            "type": "binpack"
        }
]
```

Turn on Availability Zone rebalancing

You need to enable Availability Zone rebalancing for new and existing services.

You can enable and disable Availability Zone rebalancing using the console, APIs, AWS CLI, and AWS CloudFormation.

The default behavior of `AvailabilityZoneRebalancing` differs between create and update requests:

- For create service requests, when when no value is specified for `AvailabilityZoneRebalancing`, Amazon ECS defaults the value to to `ENABLED`.
- For update service requests, when no value is specified for `AvailabilityZoneRebalancing`, Amazon ECS defaults to the existing service's `AvailabilityZoneRebalancing` value. If the service never had an `AvailabilityZoneRebalancing` value set, Amazon ECS treats this as `DISABLED`.

Service type	API	Console	CLI
Existing	UpdateService	Updating an Amazon ECS service	update-service
New	CreateService	Creating an Amazon ECS rolling update deployment	create-service

The following example shows how to enable service rebalancing when creating a new service:

```
aws ecs create-service \
    --cluster my-cluster \
    --service-name my-service \
    --task-definition my-task-definition:1 \
    --desired-count 6 \
    --availability-zone-rebalancing ENABLED
```

Track Amazon ECS Availability Zone rebalancing

You can verify if Availability Zone rebalancing is enabled for a service in the console or by calling `describe-services`. The following example can be used to see the status with the CLI.

The response will be either `ENABLED` or `DISABLED`.

```
aws ecs describe-services \
    --services service-name \
    --cluster cluster-name \
    --query services[0].availabilityZoneRebalancing
```

Service events

Amazon ECS sends service action events to help you understand the Availability Zone rebalancing lifecycle.

Event	Scenario	Type	Learn more
SERVICE_REBALANCING_STARTED	Amazon ECS starts an Availability Zone rebalancing operation	INFO	service (*service-name*) is not AZ balanced with *number-tasks* tasks in *Availability Zone 1*, *number-tasks* in *Availability Zone 2*, and *number-tasks* in *Availability Zone 3*. AZ Rebalancing in progress.
SERVICE_REBALANCING_COMPLETED	The Availability Zone rebalancing operation completes	INFO	service (*service-name*) is AZ balanced with *number-tasks* tasks in

Event	Scenario	Type	Learn more
			Availability Zone 1, *number-tasks* tasks in *Availability Zone 2* , and *number-tasks* tasks in *Availability Zone 3* .
TASKS_STARTED	Amazon ECS successfully starts tasks as part of the Availability Zone rebalancing operation	INFO	*service-name* has started *number-tasks* tasks in *Availability Zone* to AZ Rebalance: *task-ids*.
TASKS_STOPPED	Amazon ECS successfully stops tasks as part of the Availability Zone rebalancing operation	INFO	*service-name* has stopped *number-tasks* running tasks in *Availability Zone* due to AZ rebalancing: *task-id*.

Event	Scenario	Type	Learn more
SERVICE_TASK_PLACEMENT_FAILURE	Amazon ECS failed to start a task as part of the Availability Zone rebalancing operation	ERROR	For the EC2 launch type, see service (*service-name*) is unable to place a task in *Availability Zone* because no container instance met all of its requirements. For the Fargate launch type, see service (*service-name*) is unable to place a task in *Availability Zone*.
TASKSET_SCALE_IN_FAILURE_BY_TASK_PROTECTION	The Availability Zone rebalancing operation is blocked because task protection is in use.	INFO	service (*service-name*) was unable to AZ Rebalance because *task-set-name* was unable to scale in due to *reason*.
SERVICE_REBALANCING_STOPPED	The Availability Zone rebalancing operation stopped. Amazon ECS sends additional events which provide more information.	INFO	service (*service-name*) stopped AZ Rebalancing.

Task state change events

Amazon ECS sends a task state change event (`START`) for each task that it starts as part of the rebalancing process.

Amazon ECS sends a task state change event (`STOPPED`) event for each task that it stops as part of the rebalancing process. The reason is set to `Availability-zone rebalancing initiated by (deployment ecs-svc/deployment-id)`.

For more information about the events, see Amazon ECS task state change events.

Troubleshooting service rebalancing

If you encounter issues with service rebalancing, consider the following troubleshooting steps:

Rebalancing doesn't start

Verify that:

- Service rebalancing is enabled for your service
- Your service uses a supported configuration (see the section called "Considerations for configuring Availability Zone rebalancing")
- Your service has reached a steady state

Task placement failures during rebalancing

If you see `SERVICE_TASK_PLACEMENT_FAILURE` events:

- For EC2 launch type: Check if you have container instances available in the target Availability Zone
- For Fargate launch type: Check if there are resource constraints or service quotas limiting task placement
- Review your task placement constraints to ensure they're not preventing proper task distribution

Rebalancing stops unexpectedly

If you see `SERVICE_REBALANCING_STOPPED` events:

- Check for task protection that might be blocking the operation
- Look for concurrent service deployments that could interrupt rebalancing
- Review service events for additional information about why rebalancing stopped

Best practices for service rebalancing

Follow these best practices to get the most out of service rebalancing:

- **Monitor rebalancing operations** - Set up CloudWatch alarms to monitor service events related to rebalancing to quickly identify any issues.
- **Consider performance impact** - Be aware that rebalancing operations may temporarily increase resource usage as new tasks are started before old ones are stopped.
- **Use task protection strategically** - If you have critical tasks that shouldn't be terminated during rebalancing, consider using task protection.
- **Plan for EC2 capacity** - For EC2 launch type, ensure you have sufficient container instances across all Availability Zones to support effective rebalancing.
- **Test rebalancing behavior** - Before relying on rebalancing in production, test how your services behave during rebalancing operations in a non-production environment.

Use load balancing to distribute Amazon ECS service traffic

Your service can optionally be configured to use Elastic Load Balancing to distribute traffic evenly across the tasks in your service.

> **Note**
>
> When you use tasks sets, all the tasks in the set must all be configured to use Elastic Load Balancing or to not use Elastic Load Balancing.

Amazon ECS services hosted on AWS Fargate support the Application Load Balancers, Network Load Balancers, and Gateway Load Balancers. Use the following table to learn about what type of load balancer to use.

Load Balancer type	Use in these cases	
Application Load Balancer	Route HTTP/HTTPS (or layer 7) traffic. Application Load Balancers offer several features that	

Load Balancer type	Use in these cases	
	make them attractive for use with Amazon ECS services: • Each service can serve traffic from multiple load balancers and expose multiple load balanced ports by specifying multiple target groups. • They are supported by tasks hosted on both Fargate and EC2 instances. • Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance). • Application Load Balancers support path-based routing and priority rules (so that multiple services can use the same listener port on a single Application Load Balancer).	
Network Load Balancer	Route TCP or UDP (or layer 4) traffic.	

Load Balancer type	Use in these cases	
Gateway Load Balancer	Route TCP or UDP (or layer 4) traffic. Use virtual appliances, such as firewalls, intrusion detection and prevention systems, and deep packet inspection systems.	

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of these latest features, unless your service requires a feature that is only available with Network Load Balancers or Gateway Load Balancers. For more information about Elastic Load Balancing and the differences between the load balancer types, see the Elastic Load Balancing User Guide.

With your load balancer, you pay only for what you use. For more information, see Elastic Load Balancing pricing.

Optimize load balancer health check parameters for Amazon ECS

Load balancers route requests only to the healthy targets in the Availability Zones for the load balancer. Each target is registered to a target group. The load balancer checks the health of each target, using the target group health check settings. After you register the target, it must pass one health check to be considered healthy. Amazon ECS monitors the load balancer. The load balancer periodically sends health checks to the Amazon ECS container. The Amazon ECS agent monitors, and waits for the load balancer to report on the container health. It does this before it considers the container to be in a healthy status.

Two Elastic Load Balancing health check parameters affect deployment speed:

- Health check interval: Determines the approximate amount of time, in seconds, between health checks of an individual container. By default, the load balancer checks every 30 seconds.

  This parameter is named:

  - `HealthCheckIntervalSeconds` in the Elastic Load Balancing API
  - **Interval** on the Amazon EC2 console

- Healthy threshold count: Determines the number of consecutive health check successes required before considering an unhealthy container healthy. By default, the load balancer requires five passing health checks before it reports that the target container is healthy.

  This parameter is named:

  - `HealthyThresholdCount` in the Elastic Load Balancing API
  - **Healthy threshold** on the Amazon EC2 console

Important: For newly registered targets, only a single successful health check is required to consider the target healthy, regardless of the healthy threshold count setting. The healthy threshold count only applies when a target is transitioning from an unhealthy state back to a healthy state.

With the default settings, if a target becomes unhealthy and then recovers, the total time to determine the health of a container is two minutes and 30 seconds (`30 seconds * 5 = 150 seconds`).

You can speed up the health-check process if your service starts up and stabilizes in under 10 seconds. To speed up the process, reduce the health check interval and the healthy threshold count.

- `HealthCheckIntervalSeconds` (Elastic Load Balancing API name) or **Interval** (Amazon EC2 console name): 5
- `HealthyThresholdCount` (Elastic Load Balancing API name) or **Healthy threshold** (Amazon EC2 console name): 2

With this setting, the health-check process takes 10 seconds compared to the default of two minutes and 30 seconds.

For more information about the Elastic Load Balancing health check parameters, see [Health checks for your target groups]() in the *Elastic Load Balancing User Guide*.

Optimize load balancer connection draining parameters for Amazon ECS

To allow for optimization, clients maintain a keep alive connection to the container service. This allows subsequent requests from that client to reuse the existing connection. When you want to stop traffic to a container, you notify the load balancer. The load balancer periodically checks to see if the client closed the keep alive connection. The Amazon ECS agent monitors the load balancer, and waits for the load balancer to report that the keep alive connection is closed (the target is in an `UNUSED` state).

The amount of time that the load balancer waits to move the target to the `UNUSED` state is the deregistration delay. You can configure the following load balancer parameter to speed up your deployments.

- `deregistration_delay.timeout_seconds`: 300 (default)

When you have a service with a response time that's under 1 second, set the parameter to the following value to have the load balancer only wait 5 seconds before it breaks the connection between the client and the back-end service:

- `deregistration_delay.timeout_seconds`: 5

> **Note**
>
> Do not set the value to 5 seconds when you have a service with long-lived requests, such as slow file uploads or streaming connections.

SIGTERM responsiveness

Amazon ECS first sends a SIGTERM signal to the task to notify the application needs to finish and shut down. Then, Amazon ECS sends a SIGKILL message. When applications ignore the SIGTERM, the Amazon ECS service must wait to send the SIGKILL signal to terminate the process.

The amount of time that Amazon ECS waits to send the SIGKILL message is determined by the following Amazon ECS agent option:

- `ECS_CONTAINER_STOP_TIMEOUT`: 30 (default)

  For more information about the container agent parameter, see [Amazon ECS Container Agent](#) on GitHub.

To speed up the waiting period, set the Amazon ECS agent parameter to the following value:

- `ECS_CONTAINER_STOP_TIMEOUT`: 2

If your application takes more than 1 second, multiply the value by 2 and use that number as the value.

In this case, the Amazon ECS waits 2 seconds for the container to shut down, and then Amazon ECS sends a SIGKILL message when the application didn't stop.

You can also modify the application code to trap the SIGTERM signal and react to it. The following is example in JavaScript:

```
process.on('SIGTERM', function() {
  server.close();
})
```

This code causes the HTTP server to stop listening for any new requests, finish answering any in-flight requests, and then the Node.js process terminates because the event loop has nothing to do. Given this, if it takes the process only 500 ms to finish its in-flight requests, it terminates early without having to wait out the stop timeout and get sent a SIGKILL.

Use an Application Load Balancer for Amazon ECS

An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports on each container instance in your cluster. Application Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task launches, the NGINX container is registered with the Application Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see the User Guide for Application Load Balancers.

For information about the best practices for setting parameters to speed up you deployments see:

- Optimize load balancer health check parameters for Amazon ECS
- Optimize load balancer connection draining parameters for Amazon ECS

Consider the following when using Application Load Balancers with Amazon ECS:

- Amazon ECS requires the service-linked IAM role which provides the permissions needed to register and deregister targets with your load balancer when tasks are created and stopped. For more information, see Using service-linked roles for Amazon ECS.
- Target group must have the IP address type set to IPv4.

  Although tasks support IPv6, service integration with IPv6 Elastic Load Balancing isn't supported.
- For services with tasks using the `awsvpc` network mode, when you create a target group for your service, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
- If your service requires access to multiple load balanced ports, such as port 80 and port 443 for an HTTP/HTTPS service, you can configure two listeners. One listener is responsible for HTTPS that forwards the request to the service, and another listener that is responsible for redirecting HTTP requests to the appropriate HTTPS port. For more information, see Create a listener to your Application Load Balancer in the *User Guide for Application Load Balancers*.
- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
- After you create a service, the load balancer configuration can't be changed from the AWS Management Console. You can use the AWS Copilot, AWS CloudFormation, AWS CLI or SDK to modify the load balancer configuration for the `ECS` rolling deployment controller only, not AWS CodeDeploy blue/green or external. When you add, update, or remove a load balancer configuration, Amazon ECS starts a new deployment with the updated Elastic Load Balancing configuration. This causes tasks to register to and deregister from load balancers. We recommend that you verify this on a test environment before you update the Elastic Load Balancing configuration. For information about how to modify the configuration, see UpdateService in the *Amazon Elastic Container Service API Reference*.
- If a service task fails the load balancer health check criteria, the task is stopped and restarted. This process continues until your service reaches the number of desired running tasks.
- If you are experiencing problems with your load balancer-enabled services, see Troubleshooting service load balancers in Amazon ECS.
- When using `instance` target type, your tasks and load balancer must be in the same VPC. When using `ip` target type, cross-VPC connectivity is supported.
- Use a unique target group for each service.

Using the same target group for multiple services might lead to issues during service deployments.

For information about how to create an Application Load Balancer, see Create an Application Load Balancer in *Application Load Balancers*

Use a Network Load Balancer for Amazon ECS

A Network Load Balancer makes routing decisions at the transport layer (TCP/SSL). It can handle millions of requests per second. After the load balancer receives a connection, it selects a target from the target group for the default rule using a flow hash routing algorithm. It attempts to open a TCP connection to the selected target on the port specified in the listener configuration. It forwards the request without modifying the headers. Network Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task is launched, the NGINX container is registered with the Network Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see the User Guide for Network Load Balancers.

For information about the best practices for setting parameters to speed up you deployments see:

- Optimize load balancer health check parameters for Amazon ECS
- Optimize load balancer connection draining parameters for Amazon ECS

Consider the following when using Network Load Balancers with Amazon ECS:

- Amazon ECS requires the service-linked IAM role which provides the permissions needed to register and deregister targets with your load balancer when tasks are created and stopped. For more information, see Using service-linked roles for Amazon ECS.
- You cannot attach more than five target groups to a service.
- For services with tasks using the `awsvpc` network mode, when you create a target group for your service, you must choose `ip` as the target type, not `instance`. This is because tasks that use

the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
- After you create a service, the load balancer configuration can't be changed from the AWS Management Console. You can use the AWS Copilot, AWS CloudFormation, AWS CLI or SDK to modify the load balancer configuration for the ECS rolling deployment controller only, not AWS CodeDeploy blue/green or external. When you add, update, or remove a load balancer configuration, Amazon ECS starts a new deployment with the updated Elastic Load Balancing configuration. This causes tasks to register to and deregister from load balancers. We recommend that you verify this on a test environment before you update the Elastic Load Balancing configuration. For information about how to modify the configuration, see UpdateService in the *Amazon Elastic Container Service API Reference*.
- If a service task fails the load balancer health check criteria, the task is stopped and restarted. This process continues until your service reaches the number of desired running tasks.
- When you use a Gateway Load Balancer configured with IP addresses as targets and Client IP Preservation off, requests are seen as coming from the Gateway Load Balancers private IP address. This means that services behind an Gateway Load Balancer are effectively open to the world as soon as you allow incoming requests and health checks in the target security group.
- For Fargate tasks, you must use platform version `1.4.0` (Linux) or `1.0.0` (Windows).
- If you are experiencing problems with your load balancer-enabled services, see Troubleshooting service load balancers in Amazon ECS.
- When using `instance` target type, your tasks and load balancer must be in the same VPC. When using `ip` target type, cross-VPC connectivity is supported.
- The Network Load Balancer client IP address preservation is compatible with Fargate targets.
- Use a unique target group for each service.

  Using the same target group for multiple services might lead to issues during service deployments.

For information about how to create a Network Load Balancer, see Create a Network Load Balancer in *Network Load Balancers*

> **Important**
>
> If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
>
> You cannot register instances by instance ID if they have the following instance types: C1, CC1, CC2, CG1, CG2, CR1, G1, G2, HI1, HS1, M1, M2, M3, and T1. You can register instances of these types by IP address.

Use a Gateway Load Balancer for Amazon ECS

A Gateway Load Balancer operates at the third layer of the Open Systems Interconnection (OSI) model, the network layer. It listens for all IP packets across all ports and forwards traffic to the target group that's specified in the listener rule. It maintains stickiness of flows to a specific target appliance using 5-tuple (for TCP/UDP flows) or 3-tuple (for non-TCP/UDP flows). For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task is launched, the NGINX container is registered with the Gateway Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see What is a Gateway Load Balancer in *Gateway Load Balancers*.

For information about the best practices for setting parameters to speed up you deployments see:

- Optimize load balancer health check parameters for Amazon ECS
- Optimize load balancer connection draining parameters for Amazon ECS

Consider the following when using Gateway Load Balancers with Amazon ECS:

- Amazon ECS requires the service-linked IAM role which provides the permissions needed to register and deregister targets with your load balancer when tasks are created and stopped. For more information, see Using service-linked roles for Amazon ECS.

- For services with tasks using the `awsvpc` network mode, when you create a target group for your service, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
- For services with tasks that use a network mode other than `awsvpc`, Gateway Load Balancers are not supported.
- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
- After you create a service, the load balancer configuration can't be changed from the AWS Management Console. You can use the AWS Copilot, AWS CloudFormation, AWS CLI or SDK to modify the load balancer configuration for the ECS rolling deployment controller only, not AWS CodeDeploy blue/green or external. When you add, update, or remove a load balancer configuration, Amazon ECS starts a new deployment with the updated Elastic Load Balancing configuration. This causes tasks to register to and deregister from load balancers. We recommend that you verify this on a test environment before you update the Elastic Load Balancing configuration. For information about how to modify the configuration, see UpdateService in the *Amazon Elastic Container Service API Reference*.
- If a service task fails the load balancer health check criteria, the task is stopped and restarted. This process continues until your service reaches the number of desired running tasks.
- When you use a Gateway Load Balancer configured with IP addresses as targets, requests are seen as coming from the Gateway Load Balancers private IP address. This means that services behind an Gateway Load Balancer are effectively open to the world as soon as you allow incoming requests and health checks in the target security group.
- For Fargate tasks, you must use platform version `1.4.0` (Linux) or `1.0.0` (Windows).
- If you are experiencing problems with your load balancer-enabled services, see Troubleshooting service load balancers in Amazon ECS.
- When using `instance` target type, your tasks and load balancer must be in the same VPC. When using `ip` target type, cross-VPC connectivity is supported.
- Use a unique target group for each service.

  Using the same target group for multiple services might lead to issues during service deployments.

For information about how to create a Gateway Load Balancer, see Getting started with Gateway Load Balancers in *Gateway Load Balancers*

Important

If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

You cannot register instances by instance ID if they have the following instance types: C1, CC1, CC2, CG1, CG2, CR1, G1, G2, HI1, HS1, M1, M2, M3, and T1. You can register instances of these types by IP address.

Registering multiple target groups with an Amazon ECS service

Your Amazon ECS service can serve traffic from multiple load balancers and expose multiple load balanced ports when you specify multiple target groups in a service definition.

To create a service specifying multiple target groups, you must create the service using the Amazon ECS API, SDK, AWS CLI, or an AWS CloudFormation template. After the service is created, you can view the service and the target groups registered to it with the AWS Management Console. You must use `UpdateService` to modify the load balancer configuration of an existing service.

Multiple target groups can be specified in a service definition using the following format. For the full syntax of a service definition, see Service definition template.

```
"loadBalancers":[
   {
  
 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"container_name",
      "containerPort":container_port
   },
   {
  
 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"container_name",
      "containerPort":container_port
   }
]
```

Considerations

The following should be considered when you specify multiple target groups in a service definition.

- For services that use an Application Load Balancer or Network Load Balancer, you cannot attach more than five target groups to a service.
- Specifying multiple target groups in a service definition is only supported under the following conditions:
  - The service must use either an Application Load Balancer or Network Load Balancer.
  - The service must use the (ECS) deployment controller type. This can be either the Amazon ECS native/blue green deployment, or the rolling update deployment.
- Specifying multiple target groups is supported for services containing tasks using both the Fargate and EC2 launch types.
- When creating a service that specifies multiple target groups, the Amazon ECS service-linked role must be created. The role is created by omitting the `role` parameter in API requests, or the `Role` property in AWS CloudFormation. For more information, see Using service-linked roles for Amazon ECS.

Example service definitions

Following are a few example use cases for specifying multiple target groups in a service definition. For the full syntax of a service definition, see Service definition template.

Having separate load balancers for internal and external traffic

In the following use case, a service uses two separate load balancers, one for internal traffic and a second for internet-facing traffic, for the same container and port.

```
"loadBalancers":[
   //Internal ELB
   {
 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"nginx",
      "containerPort":8080
   },
   //Internet-facing ELB
```

```
    {

  "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
       "containerName":"nginx",
       "containerPort":8080
    }
]
```

Exposing multiple ports from the same container

In the following use case, a service uses one load balancer but exposes multiple ports from the same container. For example, a Jenkins container might expose port 8080 for the Jenkins web interface and port 50000 for the API.

```
"loadBalancers":[
    {

  "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
       "containerName":"jenkins",
       "containerPort":8080
    },
    {

  "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
       "containerName":"jenkins",
       "containerPort":50000
    }
]
```

Exposing ports from multiple containers

In the following use case, a service uses one load balancer and two target groups to expose ports from separate containers.

```
"loadBalancers":[
    {

  "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
```

```
        "containerName":"webserver",
        "containerPort":80
    },
    {
        
 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
        "containerName":"database",
        "containerPort":3306
    }
]
```

Automatically scale your Amazon ECS service

Automatic scaling is the ability to increase or decrease the desired number of tasks in your Amazon ECS service automatically. Amazon ECS leverages the Application Auto Scaling service to provide this functionality. For more information, see the Application Auto Scaling User Guide.

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. For more information, see Amazon ECS service utilization metrics. You can use these and other CloudWatch metrics to scale out your service (add more tasks) to deal with high demand at peak times, and to scale in your service (run fewer tasks) to reduce costs during periods of low utilization.

Amazon ECS Service Auto Scaling supports the following types of automatic scaling:

- Use a target metric to scale Amazon ECS services— Increase or decrease the number of tasks that your service runs based on a target value for a specific metric. This is similar to the way that your thermostat maintains the temperature of your home. You select temperature and the thermostat does the rest.
- Use predefined increments based on CloudWatch alarms to scale Amazon ECS services— Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.
- Use scheduled actions to scale Amazon ECS services—Increase or decrease the number of tasks that your service runs based on the date and time.
- Use historical patterns to scale Amazon ECS services with predictive scaling—Increase or decrease the number of tasks that your service runs based on historical load data analytics to detect daily or weekly patterns in traffic flows.

Considerations

When using scaling policies, consider the following:

- Amazon ECS sends metrics in 1-minute intervals to CloudWatch. Metrics are not available until the clusters and services send the metrics to CloudWatch, and you cannot create CloudWatch alarms for metrics that do not exist.
- The scaling policies support a cooldown period. This is the number of seconds to wait for a previous scaling activity to take effect.
  - For scale-out events, the intention is to continuously (but not excessively) scale out. After Service Auto Scaling successfully scales out using a scaling policy, it starts to calculate the cooldown time. The scaling policy won't increase the desired capacity again unless either a larger scale out is initiated or the cooldown period ends. While the scale-out cooldown period is in effect, the capacity added by the initiating scale-out activity is calculated as part of the desired capacity for the next scale-out activity.
  - For scale-in events, the intention is to scale in conservatively to protect your application's availability, so scale-in activities are blocked until the cooldown period has expired. However, if another alarm initiates a scale-out activity during the scale-in cooldown period, Service Auto Scaling scales out the target immediately. In this case, the scale-in cooldown period stops and doesn't complete.
- The service scheduler respects the desired count at all times, but as long as you have active scaling policies and alarms on a service, Service Auto Scaling could change a desired count that was manually set by you.
- If a service's desired count is set below its minimum capacity value, and an alarm initiates a scale-out activity, Service Auto Scaling scales the desired count up to the minimum capacity value and then continues to scale out as required, based on the scaling policy associated with the alarm. However, a scale-in activity does not adjust the desired count, because it is already below the minimum capacity value.
- If a service's desired count is set above its maximum capacity value, and an alarm initiates a scale in activity, Service Auto Scaling scales the desired count out to the maximum capacity value and then continues to scale in as required, based on the scaling policy associated with the alarm. However, a scale-out activity does not adjust the desired count, because it is already above the maximum capacity value.
- During scaling activities, the actual running task count in a service is the value that Service Auto Scaling uses as its starting point, as opposed to the desired count. This is what processing capacity is supposed to be. This prevents excessive (runaway) scaling that might not be satisfied,

for example, if there aren't enough container instance resources to place the additional tasks. If the container instance capacity is available later, the pending scaling activity may succeed, and then further scaling activities can continue after the cooldown period.

- If you want your task count to scale to zero when there's no work to be done, set a minimum capacity of 0. With target tracking scaling policies, when actual capacity is 0 and the metric indicates that there is workload demand, Service Auto Scaling waits for one data point to be sent before scaling out. In this case, it scales out by the minimum possible amount as a starting point and then resumes scaling based on the actual running task count.
- Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. This behavior does not apply to Amazon ECS services using the external deployment controller. For more information, see Service auto scaling and deployments.
- You have several Application Auto Scaling options for Amazon ECS tasks. Target tracking is the easiest mode to use. With it, all you need to do is set a target value for a metric, such as CPU average utilization. Then, the auto scaler automatically manages the number of tasks that are needed to attain that value. With step scaling you can more quickly react to changes in demand, because you define the specific thresholds for your scaling metrics, and how many tasks to add or remove when the thresholds are crossed. And, more importantly, you can react very quickly to changes in demand by minimizing the amount of time a threshold alarm is in breach.

For more information about best practices for service auto scaling, see Optimizing Amazon ECS service auto scaling.

Service auto scaling and deployments

Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. This behavior does not apply to Amazon ECS services using the external deployment controller. If you want to suspend scale-out processes while deployments are in progress, take the following steps.

1. Call the describe-scalable-targets command, specifying the resource ID of the service associated with the scalable target in Application Auto Scaling (Example: `service/default/sample-webapp`). Record the output. You will need it when you call the next command.

2. Call the [register-scalable-target](https://docs.aws.amazon.com/cli/latest/reference/application-autoscaling/register-scalable-target.html) command, specifying the resource ID, namespace, and scalable dimension. Specify `true` for both `DynamicScalingInSuspended` and `DynamicScalingOutSuspended`.
3. After deployment is complete, you can call the [register-scalable-target](https://docs.aws.amazon.com/cli/latest/reference/application-autoscaling/register-scalable-target.html) command to resume scaling.

For more information, see Suspending and resuming scaling for Application Auto Scaling.

Use a target metric to scale Amazon ECS services

With target tracking scaling policies, you select a metric and set a target value. Amazon ECS Service Auto Scaling creates and manages the CloudWatch alarms that control the scaling policy and calculates the scaling adjustment based on the metric and the target value. The scaling policy adds or removes service tasks as required to keep the metric at, or close to, the specified target value. In addition to keeping the metric close to the target value, a target tracking scaling policy also adjusts to the fluctuations in the metric due to a fluctuating load pattern and minimizes rapid fluctuations in the number of tasks running in your service.

Target tracking policies remove the need to manually define CloudWatch alarms and scaling adjustments. Amazon ECS handles this automatically based on the target you set.

Consider the following when using target tracking policies:

- A target tracking scaling policy assumes that it should perform scale out when the specified metric is above the target value. You cannot use a target tracking scaling policy to scale out when the specified metric is below the target value.
- A target tracking scaling policy does not perform scaling when the specified metric has insufficient data. It does not perform scale in because it does not interpret insufficient data as low utilization.
- You may see gaps between the target value and the actual metric data points. This is because Service Auto Scaling always acts conservatively by rounding up or down when it determines how much capacity to add or remove. This prevents it from adding insufficient capacity or removing too much capacity.
- To ensure application availability, the service scales out proportionally to the metric as fast as it can, but scales in more gradually.
- Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during

a deployment. This behavior does not apply to Amazon ECS services using the external deployment controller. For more information, see Service auto scaling and deployments.

- You can have multiple target tracking scaling policies for an Amazon ECS service, provided that each of them uses a different metric. The intention of Service Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the service if any of the target tracking policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion turned on) are ready to scale in.
- Do not edit or delete the CloudWatch alarms that Service Auto Scaling manages for a target tracking scaling policy. Service Auto Scaling deletes the alarms automatically when you delete the scaling policy.
- The `ALBRequestCountPerTarget` metric for target tracking scaling policies is not supported for the blue/green deployment type.

For more information about target tracking scaling policies, see Target tracking scaling policies in the *Application Auto Scaling User Guide*.

Create a target tracking scaling policy for Amazon ECS service auto scaling

Create a target tracking scaling policy to have Amazon ECS increase or decrease the desired task count in your service automatically. Target tracking works off of a target metric value.

Console

1. In addition to the standard IAM permissions for creating and updating services, you need additional permissions. For more information, see IAM permissions required for Amazon ECS service auto scaling.
2. Determine the metrics to use for the policy. The following metrics are available:
   - **ECSServiceAverageCPUUtilization** – The average CPU utilization the service should use.
   - **ECSServiceAverageMemoryUtilization** – Average memory utilization the service should use.
   - **ALBRequestCountPerTarget** – The average number of requests per minute that task should ideally receive.
3. Open the console at https://console.aws.amazon.com/ecs/v2.
4. On the **Clusters** page, choose the cluster.
5. On the cluster details page, in the **Services** section, and then choose the service.

The service details page appears.

6. Choose **Set the number of tasks**.
7. Under **Amazon ECS service task count**, choose **Use auto scaling**.

   The **Task count section** appears.

   a. For **Minimum number of tasks**, enter the lower limit of the number of tasks for service auto scaling to use. The desired count will not go below this count.
   b. For **Maximum**, enter the upper limit of the number of tasks for service auto scaling to use. The desired count will not go above this count.
   c. Choose **Save**.

      The policies page appears.
8. Choose **Create scaling policy**.

   The **Create policy** page appears.
9. For **Scaling policy type**, choose **Target tracking**.
10. For **Policy name**, enter the name of the policy.
11. For **Metric type**, choose your metrics from the list of options.
12. For **Target utilization**, enter the target value for the percentage of tasks that Amazon ECS should maintain. Service auto scaling scales out your capacity until the average utilization is at the target utilization, or until it reaches the maximum number of tasks you specified.
13. Under **Additional Settings**, do the following

    a. For **Scale-in cooldown period**, enter the amount of time in seconds after a scale-in activity completes before another scale-in activity can start.
    b. For **Scale-out cooldown period**, enter the amount of time in seconds to wait for a previous scale-out activity to take effect.
    c. To create only a scale-out policy, select **Disable scale-in**.
14. Choose **Create scaling policy**.

AWS CLI

1. Register your Amazon ECS service as a scalable target using the register-scalable-target command.

2. Create a scaling policy using the [put-scaling-policy](https://docs.aws.amazon.com/cli/latest/reference/application-autoscaling/put-scaling-policy.html) command.

Use predefined increments based on CloudWatch alarms to scale Amazon ECS services

With step scaling policies, you create and manage the CloudWatch alarms that invoke the scaling process. When an alarm is breached, Amazon ECS initiates the scaling policy associated with that alarm. The step scaling policy scales tasks using a set of adjustments, known as step adjustments. The size of the adjustment varies based on the magnitude of the alarm breach.

- If the breach exceeds the first threshold, Amazon ECS applies the first step adjustment.
- If the breach exceeds the second threshold, Amazon ECS applies the second step adjustment, and so on.

We strongly recommend that you use target tracking scaling policies to scale on metrics like average CPU utilization or average request count per target. Metrics that decrease when capacity increases and increase when capacity decreases can be used to proportionally scale out or in the number of tasks using target tracking. This helps ensure that Amazon ECS follows the demand curve for your applications closely.

Create a step scaling policy for Amazon ECS service auto scaling

Create a step scaling policy to have Amazon ECS increase or decrease the desired number of tasks in your service automatically. Step scaling runs based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.

Console

1. In addition to the standard IAM permissions for creating and updating services, you need additional permissions. For more information, see IAM permissions required for Amazon ECS service auto scaling.
2. Determine the metrics to use for the policy. The following metrics are available:

   - **ECSServiceAverageCPUUtilization** – The average CPU utilization the service should use.
   - **ECSServiceAverageMemoryUtilization** – Average memory utilization the service should use.
   - **ALBRequestCountPerTarget** – The average number of requests per minute that task should ideally receive.

3. Create the CloudWatch alarms for the metrics. For more information, see [Create a CloudWatch alarm based on a static threshold](#) in the *Amazon CloudWatch User Guide*.
4. Open the console at https://console.aws.amazon.com/ecs/v2.
5. On the **Clusters** page, choose the cluster.
6. On the cluster details page, in the **Services** section, and then choose the service.

   The service details page appears.
7. Choose **Set the number of tasks**.
8. Under **Amazon ECS service task count**, choose **Use auto scaling**.

   The **Task count section** appears.

   a. For **Minimum number of tasks**, enter the lower limit of the number of tasks for service auto scaling to use. The desired count will not go below this count.
   b. For **Maximum**, enter the upper limit of the number of tasks for service auto scaling to use. The desired count will not go above this count.
   c. Choose **Save**.

      The policies page appears.
9. Choose **Create scaling policy**.

   The **Create policy** page appears.
10. For **Scaling policy type**, choose **Step Scaling**.
11. Configure the scaling-out properties. Under **Steps to add tasks** do the following:

    a. For **Policy name**, enter the name of the policy.
    b. For **CloudWatch alarm name**, choose the CloudWatch alarm.
    c. For **Metric aggregation type**, choose how to compare the selected metric to the defined threshold.
    d. For A**djustment types**, choose whether the adjustment is based on a change in the number of tasks, or a change in the percentage of tasks.
    e. For **Actions to take**, enter the values for what action to take.

       Choose **Add step** to add additional actions.
12. Configure the scaling-in properties. Under **Steps to remove tasks**, do the following:

a. For **Policy name**, enter the name of the policy.

b. For **CloudWatch alarm name**, choose the CloudWatch alarm.

c. For **Metric aggregation type**, choose how to compare the selected metric to the defined threshold.

d. For **Adjustment types**, choose whether the adjustment is based on a change in the number of tasks, or a change in the percentage of tasks.

e. For **Actions to take**, enter the values for what action to take.

   Choose **Add step** to add additional actions.

13. For **Cooldown period**, enter the amount of time, in seconds, to wait for a previous scaling activity to take effect. For an add policy, this is the time after a scale-out activity that the scaling policy blocks scale-in activities and limits how many tasks can be scale out at a time. For a remove policy, this is the time after a scale-in activity that must pass before another scale-in activity can start.

14. Choose **Create scaling policy**.

AWS CLI

1. Register your Amazon ECS service as a scalable target using the register-scalable-target command.

2. Create a scaling policy using the put-scaling-policy command.

Use scheduled actions to scale Amazon ECS services

With scheduled scaling, you can set up automatic scaling for your application based on predictable load changes by creating scheduled actions that increase or decrease the number of tasks at specific times. This allows you to scale your application proactively to match predictable load changes.

These scheduled scaling actions allow you to optimize costs and performance. Your application has a sufficient number of tasks to handle the mid-week traffic peak, but does not over-provision the number of tasks at other times.

You can use scheduled scaling and scaling policies together to get the benefits of proactive and reactive approaches to scaling. After a scheduled scaling action runs, the scaling policy can continue to make decisions about whether to further scale the number of tasks. This helps you ensure that you have a sufficient number of tasks to handle the load for your application. While your application scales to match demand, current capacity must fall within the minimum and maximum number of tasks that was set by your scheduled action.

You can configure schedule scaling using the AWS CLI. For more information about scheduled scaling, see Scheduled Scaling in the *Application Auto Scaling User Guide*.

Create a scheduled action for Amazon ECS service auto scaling

Create a scheduled action to have Amazon ECS increase or decrease the number of tasks that your service runs based on the date and time.

Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, choose the service.

   The service details page appears.
4. Choose **Service auto scaling**.

   The service auto scaling page appears.
5. If you haven't configured service auto scaling, choose **Set the number of tasks**.

   The **Amazon ECS service task count** section appears.

   Under **Amazon ECS service task count**, choose **Use service auto scaling to adjust your service's desired task count**.

   The **Task count section** appears.

   a. For **Minimum number of tasks**, enter the lower limit of the number of tasks for service auto scaling to use. The desired count will not go below this count.
   b. For **Maximum**, enter the upper limit of the number of tasks for service auto scaling to use. The desired count will not go above this count.
   c. Choose **Choose Save**.

      The policies page appears.
6. Choose **Scheduled actions**, and then choose **Create**.

The **Create Scheduled action** page appears.

7. For **Action name**, enter a unique name.
8. For **Time zone**, choose a time zone.

   All of the time zones listed are from the IANA Time Zone database. For more information, see List of tz database time zones.
9. For **Start time**, enter the **Date** and **Time** the action starts.

   If you chose a recurring schedule, the start time defines when the first scheduled action in the recurring series runs.
10. For **Recurrence**, choose one of the available options.

    - To scale on a recurring schedule, choose how often Amazon ECS runs the scheduled action.
      - If you choose an option that begins with **Rate**, the cron expression is created for you.
      - If you choose **Cron**, enter a cron expression that specifies when to perform the action.
    - To scale only once, choose **Once**.
11. Under **Task adjustments**, do the following:

    - For **Minimum**, enter the minumum number of tasks the service should run.
    - For **Maximum**, enter the maximum number of tasks the service should run.
12. Choose **Create scheduled action**.

CLI

Use the AWS CLI as follows to configure scheduled scaling policies for your service. Replace each *user input placeholder* with your own information.

Example: To scale one time only

Use the following put-scheduled-action command with the `--start-time "YYYY-MM-DDThh:mm:ssZ"` and and either or both of the `--MinCapacity` and `--MaxCapacity` options.

```
aws application-autoscaling put-scheduled-action --service-namespace ecs \
  --resource-id service/my-cluster/my-service \
  --scheduled-action-name my-one-time-schedule \
  --start-time 2021-01-30T12:00:00 \
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

Example: To schedule scaling on a recurring schedule

Use the following put-scheduled-action command. Replace the *user input* with your values.

```
aws application-autoscaling put-scheduled-action --service-namespace ecs \
  --resource-id service/my-cluster/my-service \
  --scheduled-action-name my-recurring-action \
  --schedule "rate(5 hours)" \
  --start-time 2021-01-30T12:00:00 \
  --end-time 2021-01-31T22:00:00 \
  --scalable-target-action MinCapacity=3,MaxCapacity=10
```

The specified recurrence schedule runs based on the UTC time zone. To specify a different time zone, include the `--time-zone` option and the name of the IANA time zone, as in the following example.

```
--time-zone "America/New_York"
```

For more information, see List of tz database time zones.

Use historical patterns to scale Amazon ECS services with predictive scaling

Predictive scaling looks at past load data from traffic flows to analyze daily or weekly patterns. It then uses this analysis to anticipate future needs and proactively increase tasks in your service as needed.

Predictive auto scaling is most useful in the following situations.

- Cyclical traffic - Increased use of resources during regular business hours, and decreased use of resources during evenings and weekends.
- Recurring on-and-off workload patterns - Examples include batch processing, testing, or periodic data analysis.
- Applications with long initialization times - This can impact application performance during scale-out events causing of noticeable latency.

If your applications take a long time to initialize and traffic increases in a regular pattern, you should consider using predicitive scaling. It helps you scale faster by proactively increasing the number of tasks for forecasted loads, instead of using dynamic scaling policies, such as Target

Tracking or Step Scaling alone. By helping you avoid the possibility of over-provisioning the number of tasks, predictive scaling can also potentially save you money.

For example, consider an application that has high usage during business hours and low usage overnight. At the start of each business day, predictive scaling can scale-out tasks before the first influx of traffic. This helps your application maintain high availability and performance when going from a period of lower utilization to a period of higher utilization. You don't have to wait for dynamic scaling to react to changing traffic. You also don't have to spend time reviewing your application's load patterns and trying to schedule the right amount of tasks using scheduled scaling.

Predictive scaling is a service-level capability that scales the task of your service independently from the scaling of the underlying compute capacity (for example, EC2 or Fargate). For Fargate, AWS manages and automatically scales the underlying capacity based on task requirements. For EC2 capacity, you can use Auto Scaling group capacity providers to automatically scale underlying EC2 instances based on the scaling requirements of your tasks.

Contents

How predictive scaling works in Amazon ECS

Here you can learn about considerations for using predictive scaling, how it works, and what the limits are.

Considerations for using predictive scaling

- You want to make sure predictive scaling is suitable for your workload. You can check this by configuring scaling policies in **forecast only** mode and see what the console recommends. You should evaluate the forecast and recommendations before starting to use predictive scaling.
- Before predictive scaling can start forecasting, it needs at least 24 hours of historical data. The more historical data that is available, the more effective the forecast, with two weeks being ideal. You'll also need to wait 24 hours for before predictive scaling can generate new forecasts when

you delete an Amazon ECS service and create a new one. One way to speed this up is to use custom metrics to aggregate metrics across old and new Amazon ECS service.

- Choose a load metric that accurately represents the full load on your application and is the aspect of your application that's most important to scale on.
- Dynamic scaling with predictive scaling helps you follow the demand for your application closely, so you can scale in during lulls and scale out during unexpected increases in traffic. When multiple scaling policies are active, each policy determines the desired number of tasks independently, and the desired number of tasks is set to the maximum of those.
- You can use predictive scaling alongside your dynamic scaling policies, such as target tracking or step scaling, so that your applications scale based on both real-time and historic patterns. By itself, predictive scaling doesn't scale-in your tasks.
- If you use a custom role when calling the `register-scalable-target` API, you may get an error saying predictive scaling policy can only work with SLR enabled. In this case you should call `register-scalable-target` again but without role-arn. Use SLR when registering the scalable target and call the `put-scaling-policy` API.

How predictive scaling works

You use predictive scaling by creating a predictive scaling policy that specifies the CloudWatch metric to monitor and analyze. Predictive scaling must have at least 24 hours of data to start forecasting future values.

After you create the policy, predictive scaling starts analyzing metric data from up to the past 14 days to identify patterns. This analysis is used to generate the next 48 hours of requirements hourly forecasts. The latest CloudWatch data is used to update the forecast every six hours. As new data comes in, predictive scaling continuously improves the accuracy of future forecasts.

When you first enable predictive scaling, it runs in *forecast only* mode. It generates forecasts in this mode, but it doesn't scale your Amazon ECS service based on those forecasts. This means you can evaluate the accuracy and suitability of the forecast. You view forecast data by using the `GetPredictiveScalingForecast` API operation or the AWS Management Console.

When you decide to start using predictive scaling, switch the scaling policy to *forecast and scale* mode. The following occurs while in this mode.

Your Amazon ECS service is scaled at the start of each hour based on the forecast for that hour, by default. You can choose to start earlier by using the `SchedulingBufferTime` property in the

`PutScalingPolicy` API operation. This makes new tasks launch ahead of forecasted demand and gives them time to boot and become ready to handle traffic.

Maximum tasks limit

When you register Amazon ECS services for scaling, you define a maximum number of tasks that can be launched per service. By default, when scaling policies are set, they cannot increase the number of tasks higher than its maximum limit.

Alternatively, you can allow the service's maximum number of tasks to be automatically increased if the forecast approaches or exceeds the maximum number of tasks of the Amazon ECS service.

> **Warning**
>
> Use caution when allowing the maximum number of tasks to be automatically increased. This can lead to more tasks being launched than intended, if the increased maximum number of tasks isn't monitored and managed. The increased maximum number of tasks then becomes the new normal maximum number of tasks for the Amazon ECS service until you manually update it. The maximum number of tasks doesn't automatically decrease back to the original maximum.

Supported regions

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Jakarta)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)

- Asia Pacific (Tokyo)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Create a predictive scaling policy for Amazon ECS service auto scaling

Create a predictive scaling policy to have Amazon ECS increase or decrease the number of tasks that your service runs based on historical data.

> **Note**
>
> A new service needs to provide at least 24 hours of data before a forecast can be generated.

Console

1. In addition to the standard IAM permissions for creating and updating services, you need additional permissions. For more information, see IAM permissions required for Amazon ECS service auto scaling.
2. Determine the metrics to use for the policy. The following metrics are available:
   - **ECSServiceAverageCPUUtilization** – The average CPU utilization the service should use.
   - **ECSServiceAverageMemoryUtilization** – Average memory utilization the service should use.

- **ALBRequestCountPerTarget –** The average number of requests per minute that task should ideally receive.

  You can alternatively use a custom metric. You need to define the following values:

  - Load - a metric that accurately represents the full load on your application and is the aspect of your application that's most important to scale on.

  - Scaling metric - the best predictor for how much utilization is ideal for your application.

3. Open the console at https://console.aws.amazon.com/ecs/v2.
4. On the **Clusters** page, choose the cluster.
5. On the cluster details page, in the **Services** section, choose the service.

   The service details page appears.
6. Choose **Service auto scaling** and then choose **Set the number of tasks**.
7. Under **Amazon ECS service task count**, choose **Use auto scaling**.

   The **Task count section** appears.

   a. For **Minimum number of tasks**, enter the lower limit of the number of tasks for service auto scaling to use. The desired count will not go below this count.
   b. For **Maximum**, enter the upper limit of the number of tasks for service auto scaling to use. The desired count will not go above this count.
   c. Choose **Save**.

      The policies page appears.
8. Choose **Create scaling policy**.

   The **Create policy** page appears.
9. For **Scaling policy type**, choose **Predictive Scaling**.
10. For **Policy name**, enter the name of the policy.
11. For **Metric pair**, choose your metrics from the list of options.

    If you chose **Application Load Balancer request count per target**, then choose a target group in **Target group**. **Application Load Balancer request count per target** is only supported if you have attached an Application Load Balancer target group for your service.

If you chose **Custom metric pair**, choose individual metrics from the lists for **Load metric** and **Scaling metric**.

12. For **Target utilization**, enter the target value for the percentage of tasks that Amazon ECS should maintain. Service auto scaling scales out your capacity until the average utilization is at the target utilization, or until it reaches the maximum number of tasks you specified.
13. Choose **Create scaling policy**.

AWS CLI

Use the AWS CLI as follows to configure predictive scaling policies for your Amazon ECS service. Replace each *user input placeholder* with your own information.

For more information about the CloudWatch metrics you can specify, see PredictiveScalingMetricSpecification in the *Amazon EC2 Auto Scaling API Reference*.

Example 1: A predictive scaling policy with predefined memory.

The following is an example policy with a predefined memory configuration.

```
cat policy.json
{
    "MetricSpecifications": [
        {
            "TargetValue": 40,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ECSServiceMemoryUtilization"
            }
        }
    ],
    "SchedulingBufferTime": 3600,
    "MaxCapacityBreachBehavior": "HonorMaxCapacity",
    "Mode": "ForecastOnly"
}
```

The following example illustrates creating the policy by running the put-scaling-policy command with the configuration file specified.

```
aws application-autoscaling put-scaling-policy \
--service-namespace ecs \
```

```
--region us-east-1 \
--policy-name predictive-scaling-policy-example \
--resource-id service/MyCluster/test \
--policy-type PredictiveScaling \
--scalable-dimension ecs:service:DesiredCount \
--predictive-scaling-policy-configuration file://policy.json
```

If successful, this command returns the policy's ARN.

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-1:012345678912:scalingPolicy:d1d72dfe-5fd3-464f-83cf-824f16cb88b7:resource/ecs/
service/MyCluster/test:policyName/predictive-scaling-policy-example",
    "Alarms": []
}
```

Example 2: A predictive scaling policy with predefined CPU.

The following is an example policy with a predefined CPU configuration.

```
cat policy.json
{
    "MetricSpecifications": [
        {
            "TargetValue": 0.00000004,
            "PredefinedMetricPairSpecification": {
                "PredefinedMetricType": "ECSServiceCPUUtilization"
            }
        }
    ],
    "SchedulingBufferTime": 3600,
    "MaxCapacityBreachBehavior": "HonorMaxCapacity",
    "Mode": "ForecastOnly"
}
```

The following example illustrates creating the policy by running the put-scaling-policy command with the configuration file specified.

```
aws aas put-scaling-policy \
--service-namespace ecs \
--region us-east-1 \
--policy-name predictive-scaling-policy-example \
```

```
--resource-id service/MyCluster/test \
--policy-type PredictiveScaling \
--scalable-dimension ecs:service:DesiredCount \
--predictive-scaling-policy-configuration file://policy.json
```

If successful, this command returns the policy's ARN.

```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-1:012345678912:scalingPolicy:d1d72dfe-5fd3-464f-83cf-824f16cb88b7:resource/ecs/
service/MyCluster/test:policyName/predictive-scaling-policy-example",
    "Alarms": []
}
```

Evaluate your predictive scaling policies for Amazon ECS

Before you use a predictive scaling policy to scale your services, review the recommendations and other data for your policy in the Amazon ECS console. This is important because you don't want a predictive scaling policy to scale your actual capacity until you know that its predictions are accurate.

If the service is new, allow 24 hours to create the first forecast.

When AWS creates a forecast, it uses historical data. If your service doesn't have much recent historical data yet, predictive scaling might temporarily backfill the forecast with aggregates created from the currently available historical aggregates. Forecasts are backfilled for up to two weeks before a policy's creation date.

View your predictive scaling recommendations

For effective analysis, service auto scaling should have at least two predictive scaling policies to compare. (However, you can still review the findings for a single policy.) When you create multiple policies, you can evaluate a policy that uses one metric against a policy that uses a different metric. You can also evaluate the impact of different target value and metric combinations. After the predictive scaling policies are created, Amazon ECS immediately starts evaluating which policy would do a better job of scaling your group.

To view your recommendations in the Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.

2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, choose the service.

   The service details page appears.
4. Choose **Service auto scaling**.
5. Choose the predictive scaling policy, and then choose **Actions**, **Predictive Scaling**, **View recommendation**.

   You can view details about a policy along with our recommendation. The recommendation tells you whether the predictive scaling policy does a better job than not using it.

   If you're unsure whether a predictive scaling policy is appropriate for your group, review the **Availability impact** and **Cost impact** columns to choose the right policy. The information for each column tells you what the impact of the policy is.

   - **Availability impact**: Describes whether the policy would avoid negative impact to availability by provisioning enough tasks to handle the workload, compared to not using the policy.
   - **Cost impact**: Describes whether the policy would avoid negative impact on your costs by not over-provisioning tasks, compared to not using the policy. By over-provisioning too much, your services are underutilized or idle, which only adds to the cost impact.

   If you have multiple policies, then a **Best prediction** tag displays next to the name of the policy that gives the most availability benefits at lower cost. More weight is given to availability impact.
6. (Optional) To select the desired time period for recommendation results, choose your preferred value from the **Evaluation period** dropdown: **2 days**, **1 week**, or **2 weeks**. By default, the evaluation period is the last two weeks. A longer evaluation period provides more data points to the recommendation results. However, adding more data points might not improve the results if your load patterns have changed, such as after a period of exceptional demand. In this case, you can get a more focused recommendation by looking at more recent data.

> **Note**
>
> Recommendations are generated only for policies that are in **Forecast only** mode. The recommendations feature works better when a policy is in the **Forecast only** mode

throughout the evaluation period. If you start a policy in **Forecast and scale** mode and switch it to **Forecast only** mode later, the findings for that policy are likely to be biased. This is because the policy has already contributed toward the actual capacity.

Review predictive scaling monitoring graphs

In the console, you can review the forecast of the previous days, weeks, or months to visualize how well the policy performs over time. You can also use this information to evaluate the accuracy of predictions when deciding whether to let a policy scale your actual number of tasks.

To review predictive scaling monitoring graphs in the Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, choose the service.

   The service details page appears.
4. Choose **Service auto scaling**.
5. Choose the predictive scaling policy, and then choose **Actions**, **Predictive Scaling**, **View Graph**.
6. In the **Monitoring** section, you can view your policy's past and future forecasts for load and capacity against actual values. The **Load** graph shows load forecast and actual values for the load metric that you chose. The **Capacity** graph shows the number of tasks predicted by the policy. It also includes the actual number of tasks launched. The vertical line separates historical values from future forecasts. These graphs become available shortly after the policy is created.
7. (Optional) To change the amount of historical data shown in the chart, choose your preferred value from the **Evaluation period** dropdown at the top of the page. The evaluation period does not transform the data on this page in any way. It only changes the amount of historical data shown.

Compare data in the Load graph

Each horizontal line represents a different set of data points reported in one-hour intervals:

1. **Actual observed load** uses the SUM statistic for your chosen load metric to show the total hourly load in the past.

2. **Load predicted by the policy** shows the hourly load prediction. This prediction is based on the previous two weeks of actual load observations.

Compare data in the Capacity graph

Each horizontal line represents a different set of data points reported in one-hour intervals:

1. **Actual observed number of tasks** shows your Amazon ECS service actual capacity in the past, which depends on your other scaling policies and minimum group size in effect for the selected time period.
2. **Capacity predicted by the policy** shows the baseline capacity that you can expect to have at the beginning of each hour when the policy is in **Forecast and scale** mode.
3. **Inferred required number of tasks** shows the ideal number of tasks in your service to maintain the scaling metric at the target value you chose.
4. **Minimum number of tasks** shows the minimum number of tasks in your service.
5. **Maximum capacity** shows the maximum number of tasks in your service.

For the purpose of calculating the inferred required capacity, we begin by assuming that each task is equally utilized at a specified target value. In practice, the number of tasks are not equally utilized. By assuming that utilization is uniformly spread between tasks, however, we can make a likelihood estimate of the amount of capacity that is needed. The requirement for thenumber of tasks is then calculated to be inversely proportional to the scaling metric that you used for your predictive scaling policy. In other words, as the number of tasks increase, the scaling metric decreases at the same rate. For example, if the number of tasks doubles, the scaling metric must decrease by half.

The formula for the inferred required capacity:

```
sum of (actualServiceUnits*scalingMetricValue)/(targetUtilization)
```

For example, we take the `actualServiceUnits` (`10`) and the `scalingMetricValue` (`30`) for a given hour. We then take the `targetUtilization` that you specified in your predictive scaling policy (`60`) and calculate the inferred required capacity for the same hour. This returns a value of 5. This means that five is the inferred amount of capacity required to maintain capacity in direct inverse proportion to the target value of the scaling metric.

> **Note**
>
> Various levers are available for you to adjust and improve the cost savings and availability of your application.
>
> - You use predictive scaling for the baseline capacity and dynamic scaling to handle additional capacity. Dynamic scaling works independently from predictive scaling, scaling in and out based on current utilization. First, Amazon ECS calculates the recommended number of tasks for each non-scheduled scaling policy. Then, it scales based on the policy that provides the largest number of tasks.
> - To allow scale in to occur when the load decreases, your service should always have at least one dynamic scaling policy with the scale-in portion enabled.
> - You can improve scaling performance by making sure that your minimum and maximum capacity are not too restrictive. A policy with a recommended number of tasks that does not fall within the minimum and maximum capacity range will be prevented from scaling in and out.

Monitor predictive scaling metrics for Amazon ECS with CloudWatch

You can use Amazon CloudWatch to monitor your data for predictive scaling. A predictive scaling policy collects data that is used to forecast your future load. The data collected is automatically stored in CloudWatch at regular intervals and can be used to visualize how well the policy performs over time. You can also create CloudWatch alarms to notify you when performance indicators change beyond the limits that you defined.

Visualize historical forecast data

Load forecast data for a predictive scaling policy can be viewed in CloudWatch and can be useful when visualizing forecasts against other CloudWatch metrics in a single graph. You can also see trends over time by viewing a broader time range. You can access up to 15 months of historical metrics to get a better perspective on how your policy is performing.

To view historical forecast data using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics** and then **All metrics**.
3. Choose the **Application Auto Scaling** metric namespace.

4. Choose **Predictive Scaling Load Forecasts**.
5. In the search field, enter the name of the predictive scaling policy or the name of the Amazon ECS service group, and then press Enter to filter the results.
6. To graph a metric, select the check box next to the metric. To change the name of the graph, choose the pencil icon. To change the time range, select one of the predefined values or choose **custom**. For more information, see Graphing a metric in the *Amazon CloudWatch User Guide*.
7. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value, and then choose a different statistic. Although you can choose any statistic for each metric, not all statistics are useful for **PredictiveScalingLoadForecast** metrics. For example, the **Average**, **Minimum**, and **Maximum** statistics are useful, but the **Sum** statistic is not.
8. To add another metric to the graph, under **Browse**, choose **All**, find the specific metric, and then select the check box next to it. You can add up to 10 metrics.
9. (Optional) To add the graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Create accuracy metrics using metric math

With metric math, you can query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series on the CloudWatch console and add them to dashboards. For more information about metric math, see Using metric math in the *Amazon CloudWatch User Guide*.

Using metric math, you can graph the data that service auto scaling generates for predictive scaling in different ways. This helps you monitor policy performance over time, and helps you understand whether your combination of metrics can be improved.

For example, you can use a metric math expression to monitor the mean absolute percentage error (MAPE). The MAPE metric helps monitor the difference between the forecasted values and the actual values observed during a given forecast window. Changes in the value of MAPE can indicate whether the policy's performance is degrading over time as the nature of your application changes. An increase in MAPE signals a wider gap between the forecasted values and the actual values.

Example: Metric math expression

To get started with this type of graph, you can create a metric math expression like the one shown in the following example.

Instead of a single metric, there is an array of metric data query structures for `MetricDataQueries`. Each item in `MetricDataQueries` gets a metric or performs a math expression. The first item, `e1`, is the math expression. The designated expression sets the `ReturnData` parameter to `true`, which ultimately produces a single time series. For all other metrics, the `ReturnData` value is `false`.

In the example, the designated expression uses the actual and forecasted values as input and returns the new metric (MAPE). `m1` is the CloudWatch metric that contains the actual load values (assuming CPU utilization is the load metric that was originally specified for the policy named `my-predictive-scaling-policy`). `m2` is the CloudWatch metric that contains the forecasted load values. The math syntax for the MAPE metric is as follows:

Average of (abs ((Actual - Forecast)/(Actual)))

Visualize your accuracy metrics and set alarms

To visualize the accuracy metric data, select the **Metrics** tab in the CloudWatch console. You can graph the data from there. For more information, see Adding a math expression to a CloudWatch graph in the *Amazon CloudWatch User Guide*.

You can also set an alarm on a metric that you're monitoring from the **Metrics** section. While on the **Graphed metrics** tab, select the **Create alarm** icon under the **Actions** column. The **Create alarm** icon is represented as a small bell. For more information and notification options, see Creating a CloudWatch alarm based on a metric math expression and Notifying users on alarm changes in the *Amazon CloudWatch User Guide*.

Alternatively, you can use GetMetricData and PutMetricAlarm to perform calculations using metric math and create alarms based on the output.

Use scheduled actions to override forecast values for Amazon ECS

Sometimes, you might have additional information about your future application requirements that the forecast calculation is unable to take into account. For example, forecast calculations might underestimate the tasks needed for an upcoming marketing event. You can use scheduled actions to temporarily override the forecast during future time periods. The scheduled actions can run on a recurring basis, or at a specific date and time when there are one-time demand fluctuations.

For example, you can create a scheduled action with a higher number of tasks than what is forecasted. At runtime, Amazon ECS updates the minimum number of tasks in your service. Because predictive scaling optimizes for the number of tasks, a scheduled action with a minimum number of tasks that is higher than the forecast values is honored. This prevents the number of tasks from being less than expected. To stop overriding the forecast, use a second scheduled action to return the minimum number of tasks to its original setting.

The following procedure outlines the steps for overriding the forecast during future time periods.

Topics

> **Important**
>
> This topic assumes that you are trying to override the forecast to scale to a higher capacity than what is forecasted. If you need to temporarily decrease the number of tasks without interference from a predictive scaling policy, use *forecast only* mode instead. While in forecast only mode, predictive scaling will continue to generate forecasts, but it will not automatically increase the number of tasks. You can then monitor resource utilization and manually decrease the number of tasks as needed.

Step 1: (Optional) Analyze time series data

Start by analyzing the forecast time series data. This is an optional step, but it is helpful if you want to understand the details of the forecast.

1. **Retrieve the forecast**

   After the forecast is created, you can query for a specific time period in the forecast. The goal of the query is to get a complete view of the time series data for a specific time period.

   Your query can include up to two days of future forecast data. If you have been using predictive scaling for a while, you can also access your past forecast data. However, the maximum time duration between the start and end time is 30 days.

   To get the forecast using the get-predictive-scaling-forecast AWS CLI command, provide the following parameters in the command:

   - Enter the name of the cluster name in the `resource-id` parameter.
   - Enter the name of the policy in the `--policy-name` parameter.

- Enter the start time in the `--start-time` parameter to return only forecast data for after or at the specified time.
- Enter the end time in the `--end-time` parameter to return only forecast data for before the specified time.

```
aws application-autoscaling get-predictive-scaling-forecast \
    --service-namespace ecs \
    --resource-id service/MyCluster/test \
    --policy-name cpu40-predictive-scaling-policy \
    --scalable-dimension ecs:service:DesiredCount \
    --start-time "2021-05-19T17:00:00Z" \
    --end-time "2021-05-19T23:00:00Z"
```

If successful, the command returns data similar to the following example.


```
{
    "LoadForecast": [
        {
            "Timestamps": [
                "2021-05-19T17:00:00+00:00",
                "2021-05-19T18:00:00+00:00",
                "2021-05-19T19:00:00+00:00",
                "2021-05-19T20:00:00+00:00",
                "2021-05-19T21:00:00+00:00",
                "2021-05-19T22:00:00+00:00",
                "2021-05-19T23:00:00+00:00"
            ],
            "Values": [
                153.0655799339254,
                128.8288551285919,
                107.1179447150675,
                197.3601844551528,
                626.4039934516954,
                596.9441277518481,
                677.9675713779869
            ],
            "MetricSpecification": {
                "TargetValue": 40.0,
                "PredefinedMetricPairSpecification": {
                    "PredefinedMetricType": "ASGCPUUtilization"
                }
```

```
            }
        }
    ],
    "CapacityForecast": {
        "Timestamps": [
            "2021-05-19T17:00:00+00:00",
            "2021-05-19T18:00:00+00:00",
            "2021-05-19T19:00:00+00:00",
            "2021-05-19T20:00:00+00:00",
            "2021-05-19T21:00:00+00:00",
            "2021-05-19T22:00:00+00:00",
            "2021-05-19T23:00:00+00:00"
        ],
        "Values": [
            2.0,
            2.0,
            2.0,
            2.0,
            4.0,
            4.0,
            4.0
        ]
    },
    "UpdateTime": "2021-05-19T01:52:50.118000+00:00"
}
```

The response includes two forecasts: `LoadForecast` and `CapacityForecast`. `LoadForecast` shows the hourly load forecast. `CapacityForecast` shows forecast values for the capacity that is needed on an hourly basis to handle the forecasted load while maintaining a `TargetValue` of 40.0 (40% average CPU utilization).

2. **Identify the target time period**

   Identify the hour or hours when the one-time demand fluctuation should take place. Remember that dates and times shown in the forecast are in UTC.

Step 2: Create two scheduled actions

Next, create two scheduled actions for a specific time period when your application will have a higher than forecasted load. For example, if you have a marketing event that will drive traffic to your site for a limited period of time, you can schedule a one-time action to update the minimum capacity when it starts. Then, schedule another action to return the minimum capacity to the original setting when the event ends.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, and then choose the service.

   The service details page appears.
4. Choose **Service Auto Scaling**.

   The policies page appears.
5. Choose **Scheduled actions**, and then choose **Create**.

   The **Create Schedule action** page appears.
6. For **Action name**, enter a unique name.
7. For **Time zone**, choose a time zone.

   All of the time zones listed are from the IANA Time Zone database. For more information, see List of tz database time zones.
8. For **Start time**, enter the **Date** and **Time** the action starts.
9. For **Recurrence**, choose **Once**.
10. Under **Task adjustments**, For Minimum, enter a value less than or equal to the maximum number of tasks..
11. Choose **Create scheduled action**.

    The policies page appears.
12. Configure a second scheduled action to return the minimum number of tasks to the original setting at the end of the event. Predictive scaling can scale the number of tasks only when the value you set for **Minimum** is lower than the forecast values.

To create two scheduled actions for one-time events (AWS CLI)

To use the AWS CLI to create the scheduled actions, use the put-scheduled-update-group-action command.

For example, let's define a schedule that maintains a minimum capacity of three instances on May 19 at 5:00 PM for eight hours. The following commands show how to implement this scenario.

The first put-scheduled-update-group-action command instructs Amazon EC2 Auto Scaling to update the minimum capacity of the specified Auto Scaling group at 5:00 PM UTC on May 19, 2021.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-
start \
  --auto-scaling-group-name my-asg --start-time "2021-05-19T17:00:00Z" --minimum-
capacity 3
```

The second command instructs Amazon EC2 Auto Scaling to set the group's minimum capacity to one at 1:00 AM UTC on May 20, 2021.

```
aws autoscaling put-scheduled-update-group-action --scheduled-action-name my-event-end
 \
  --auto-scaling-group-name my-asg --start-time "2021-05-20T01:00:00Z" --minimum-
capacity 1
```

After you add these scheduled actions to the Auto Scaling group, Amazon EC2 Auto Scaling does the following:

- At 5:00 PM UTC on May 19, 2021, the first scheduled action runs. If the group currently has fewer than three instances, the group scales out to three instances. During this time and for the next eight hours, Amazon EC2 Auto Scaling can continue to scale out if the predicted capacity is higher than the actual capacity or if there is a dynamic scaling policy in effect.
- At 1:00 AM UTC on May 20, 2021, the second scheduled action runs. This returns the minimum capacity to its original setting at the end of the event.

Scaling based on recurring schedules

To override the forecast for the same time period every week, create two scheduled actions and provide the time and date logic using a cron expression.

The cron expression format consists of five fields separated by spaces: [Minute] [Hour] [Day_of_Month] [Month_of_Year] [Day_of_Week]. Fields can contain any allowed values, including special characters.

For example, the following cron expression runs the action every Tuesday at 6:30 AM. The asterisk is used as a wildcard to match all values for a field.

```
30 6 * * 2
```

See also

For more information about how to manage scheduled actions, see [Use scheduled actions to scale Amazon ECS services](#).

Advanced predictive scaling policy using custom metrics for Amazon ECS

You can use predefined or custom metrics in a predictive scaling policy. Custom metrics are useful when the predefined metrics, such as CPU, memory, etc) aren't enough to sufficiently describe your application load.

When creating a predictive scaling policy with custom metrics, you can specify other CloudWatch metrics provided by AWS. Alternatively, you can specify metrics that you define and publish yourself. You can also use metric math to aggregate and transform existing metrics into a new time series that AWS doesn't automatically track. An example is combining values in your data by calculating new sums or averages called *aggregating*. The resulting data is called an *aggregate*.

The following section contains best practices and examples of how to construct the JSON structure for the policy.

Prerequisites

To add custom metrics to your predictive scaling policy, you must have `cloudwatch:GetMetricData` permissions.

To specify your own metrics instead of the metrics that AWS provides, you must first publish your metrics to CloudWatch. For more information, see [Publishing custom metrics](#) in the *Amazon CloudWatch User Guide*.

If you publish your own metrics, make sure to publish the data points at a minimum frequency of five minutes. Data points are retrieved from CloudWatch based on the length of the period that it needs. For example, the load metric specification uses hourly metrics to measure the load on your application. CloudWatch uses your published metric data to provide a single data value for any one-hour period by aggregating all data points with timestamps that fall within each one-hour period.

Best practices

The following best practices can help you use custom metrics more effectively:

- The most useful metric for the load metric specification is a metric that represents the load on an Auto Scaling group as a whole.
- The most useful metric for the scaling metric specification to scale by is an average throughput or utilization per task metric.
- The target utilization must match the type of scaling metric. For a policy configuration that uses CPU utilization, this is a target percentage, for example.
- If these recommendations are not followed, the forecasted future values of the time series will probably be incorrect. To validate that the data is correct, you can view the forecasted values in the console. Alternatively, after you create your predictive scaling policy, inspect the `LoadForecast` objects returned by a call to the GetPredictiveScalingForecast API.
- We strongly recommend that you configure predictive scaling in forecast only mode so that you can evaluate the forecast before predictive scaling starts actively scaling.

Limitations

- You can query data points of up to 10 metrics in one metric specification.
- For the purposes of this limit, one expression counts as one metric.

Troubleshooting a predictive scaling policy with custom metrics

If an issue occurs while using custom metrics, we recommend that you do the following:

- If you encounter an issue in a blue/green deployment while using a search expression, make sure you created an search expression that's looking for a partial match and not an exact match. You should also check that the query is only finding Auto Scaling groups running in the specific application. For more information about the search expression syntax, see CloudWatch search expression syntax in the *Amazon CloudWatch User Guide*.
- The put-scaling-policy command validates an expression when you create your scaling policy. However, there's a possibility that this command might fail to identify the exact cause of the detected errors. To fix the issues, troubleshoot the errors that you receive in a response from a request to the get-metric-data command. You can also troubleshoot the expression from the CloudWatch console.
- You must specify `false` for `ReturnData` if `MetricDataQueries` specifies the SEARCH() function on its own without a math function like SUM(). This is because search expressions might return multiple time series, and a metric specification based on an expression can return only one time series.

- All metrics involved in a search expression should be of the same resolution.

Constructing the JSON for predictive scaling custom metrics with Amazon ECS

The following section contains examples for how to configure predictive scaling to query data from CloudWatch. There are two different methods to configure this option, and the method that you choose affects which format you use to construct the JSON for your predictive scaling policy. When you use metric math, the format of the JSON varies further based on the metric math being performed.

1. To create a policy that gets data directly from other CloudWatch metrics provided by AWS or metrics that you publish to CloudWatch, see [Example predictive scaling policy with custom load and scaling metrics using the AWS CLI](#).

Example predictive scaling policy with custom load and scaling metrics using the AWS CLI

To create a predictive scaling policy with custom load and scaling metrics with the AWS CLI, store the arguments for `--predictive-scaling-configuration` in a JSON file named `config.json`.

You start adding custom metrics by replacing the replaceable values in the following example with those of your metrics and your target utilization.

```
{
  "MetricSpecifications": [
    {
      "TargetValue": 50,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "scaling_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "MyUtilizationMetric",
                "Namespace": "MyNameSpace",
                "Dimensions": [
                  {
                    "Name": "MyOptionalMetricDimensionName",
                    "Value": "MyOptionalMetricDimensionValue"
                  }
                ]
```

```
            },
            "Stat": "Average"
          }
        }
      ]
    },
    "CustomizedLoadMetricSpecification": {
      "MetricDataQueries": [
        {
          "Id": "load_metric",
          "MetricStat": {
            "Metric": {
              "MetricName": "MyLoadMetric",
              "Namespace": "MyNameSpace",
              "Dimensions": [
                {
                  "Name": "MyOptionalMetricDimensionName",
                  "Value": "MyOptionalMetricDimensionValue"
                }
              ]
            },
            "Stat": "Sum"
          }
        }
      ]
    }
  }
  ]
}
```

For more information, see MetricDataQuery in the *Amazon EC2 Auto Scaling API Reference*.

Note

Following are some additional resources that can help you find metric names, namespaces, dimensions, and statistics for CloudWatch metrics:

- For information about the available metrics for AWS services, see AWS services that publish CloudWatch metrics in the *Amazon CloudWatch User Guide*.
- To get the exact metric name, namespace, and dimensions (if applicable) for a CloudWatch metric with the AWS CLI, see list-metrics.

To create this policy, run the put-scaling-policy command using the JSON file as input, as demonstrated in the following example.

```
aws application-autoscaling put-scaling-policy --policy-name my-predictive-scaling-
policy \
  --auto-scaling-group-name my-asg --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
```

If successful, this command returns the policy's Amazon Resource Name (ARN).

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-
b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-predictive-scaling-policy",
  "Alarms": []
}
```

Use metric math expressions

The following section provides information about using metric math with predictive scaling policies in your policy.

Understand metric math

If all you want to do is aggregate existing metric data, CloudWatch metric math saves you the effort and cost of publishing another metric to CloudWatch. You can use any metric that AWS provides, and you can also use metrics that you define as part of your applications.

For more information, see Using metric math in the *Amazon CloudWatch User Guide*.

If you choose to use a metric math expression in your predictive scaling policy, consider the following points:

- Metric math operations use the data points of the unique combination of metric name, namespace, and dimension keys/value pairs of metrics.
- You can use any arithmetic operator (+ - * / ^), statistical function (such as AVG or SUM), or other function that CloudWatch supports.
- You can use both metrics and the results of other math expressions in the formulas of the math expression.

- Your metric math expressions can be made up of different aggregations. However, it's a best practice for the final aggregation result to use `Average` for the scaling metric and `Sum` for the load metric.
- Any expressions used in a metric specification must eventually return a single time series.

To use metric math, do the following:

- Choose one or more CloudWatch metrics. Then, create the expression. For more information, see Using metric math in the *Amazon CloudWatch User Guide*.
- Verify that the metric math expression is valid by using the CloudWatch console or the CloudWatch GetMetricData API.

Example predictive scaling policy that combines metrics using metric math (AWS CLI)

Sometimes, instead of specifying the metric directly, you might need to first process its data in some way. For example, you might have an application that pulls work from an Amazon SQS queue, and you might want to use the number of items in the queue as criteria for predictive scaling. The number of messages in the queue does not solely define the number of instances that you need. Therefore, more work is needed to create a metric that can be used to calculate the backlog per instance.

The following is an example predictive scaling policy for this scenario. It specifies scaling and load metrics that are based on the Amazon SQS `ApproximateNumberOfMessagesVisible` metric, which is the number of messages available for retrieval from the queue. It also uses the Amazon EC2 Auto Scaling `GroupInServiceInstances` metric and a math expression to calculate the backlog per instance for the scaling metric.

```
aws application-autoscaling put-scaling-policy --policy-name my-sqs-custom-metrics-
policy \
  --policy-type PredictiveScaling \
  --predictive-scaling-configuration file://config.json
  --service-namespace ecs \
  --resource-id service/MyCluster/test \
  "MetricSpecifications": [
    {
      "TargetValue": 100,
      "CustomizedScalingMetricSpecification": {
        "MetricDataQueries": [
          {
```

```
            "Label": "Get the queue size (the number of messages waiting to be
processed)",
            "Id": "queue_size",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Get the group size (the number of running instances)",
            "Id": "running_capacity",
            "MetricStat": {
              "Metric": {
                "MetricName": "GroupInServiceInstances",
                "Namespace": "AWS/AutoScaling",
                "Dimensions": [
                  {
                    "Name": "AutoScalingGroupName",
                    "Value": "my-asg"
                  }
                ]
              },
              "Stat": "Sum"
            },
            "ReturnData": false
          },
          {
            "Label": "Calculate the backlog per instance",
            "Id": "scaling_metric",
            "Expression": "queue_size / running_capacity",
            "ReturnData": true
          }
        ]
      },
```

```
      "CustomizedLoadMetricSpecification": {
        "MetricDataQueries": [
          {
            "Id": "load_metric",
            "MetricStat": {
              "Metric": {
                "MetricName": "ApproximateNumberOfMessagesVisible",
                "Namespace": "AWS/SQS",
                "Dimensions": [
                  {
                    "Name": "QueueName",
                    "Value": "my-queue"
                  }
                ],
              },
              "Stat": "Sum"
            },
            "ReturnData": true
          }
        ]
      }
    }
  ]
}
```

The example returns the policy's ARN.

```
{
  "PolicyARN": "arn:aws:autoscaling:region:account-id:scalingPolicy:2f4f5048-d8a8-4d14-b13a-d1905620f345:autoScalingGroupName/my-asg:policyName/my-sqs-custom-metrics-policy",
  "Alarms": []
}
```

Interconnect Amazon ECS services

Applications that run in Amazon ECS tasks often need to receive connections from the internet or to connect to other applications that run in Amazon ECS services. If you need external connections from the internet, we recommend using Elastic Load Balancing. For more information about integrated load balancing, see the section called "Use load balancing to distribute service traffic".

If you need an application to connect to other applications that run in Amazon ECS services, Amazon ECS provides the following ways to do this without a load balancer:

- *Amazon ECS Service Connect*

  We recommend Service Connect, which provides Amazon ECS configuration for service discovery, connectivity, and traffic monitoring. With Service Connect, your applications can use short names and standard ports to connect to Amazon ECS services in the same cluster, other clusters, including across VPCs in the same AWS Region.

  When you use Service Connect, Amazon ECS manages all of the parts of service discovery: creating the names that can be discovered, dynamically managing entries for each task as the tasks start and stop, running an agent in each task that is configured to discover the names. Your application can look up the names by using the standard functionality for DNS names and making connections. If your application does this already, you don't need to modify your application to use Service Connect.

  You provide the complete configuration inside each service and task definition. Amazon ECS manages changes to this configuration in each service deployment, to ensure that all tasks in a deployment behave in the same way. For example, a common problem with DNS as service discovery is controlling a migration. If you change a DNS name to point to the new replacement IP addresses, it might take the maximum TTL time before all the clients begin using the new service. With Service Connect, the client deployment updates the configuration by replacing the client tasks. You can configure the deployment circuit breaker and other deployment configuration to affect Service Connect changes in the same way as any other deployment.

  For more information, see Use Service Connect to connect Amazon ECS services with short names.
- *Amazon ECS service discovery*

  Another approach for service-to-service communication is direct communication using service discovery. In this approach, you can use the AWS Cloud Map service discovery integration with Amazon ECS. Using service discovery, Amazon ECS syncs the list of launched tasks to AWS Cloud Map, which maintains a DNS hostname that resolves to the internal IP addresses of one or more tasks from that particular service. Other services in the Amazon VPC can use this DNS hostname to send traffic directly to another container using its internal IP address.

  This approach to service-to-service communication provides low latency. There are no extra components between the containers. Traffic travels directly from one container to the other container.

This approach is suitable when using the `awsvpc` network mode, where each task has its own unique IP address. Most software only supports the use of DNS `A` records, which resolve directly to IP addresses. When using the `awsvpc` network mode, the IP address for each task are an `A` record. However, if you're using `bridge` network mode, multiple containers could be sharing the same IP address. Additionally, dynamic port mappings cause the containers to be randomly assigned port numbers on that single IP address. At this point, an `A` record is no longer enough for service discovery. You must also use an `SRV` record. This type of record can keep track of both IP addresses and port numbers but requires that you configure applications appropriately. Some prebuilt applications that you use might not support `SRV` records.

Another advantage of the `awsvpc` network mode is that you have a unique security group for each service. You can configure this security group to allow incoming connections from only the specific upstream services that need to talk to that service.

The main disadvantage of direct service-to-service communication using service discovery is that you must implement extra logic to have retries and deal with connection failures. DNS records have a time-to-live (TTL) period that controls how long they are cached for. It takes some time for the DNS record to be updated and for the cache to expire so that your applications can pick up the latest version of the DNS record. So, your application might end up resolving the DNS record to point at another container that's no longer there. Your application needs to handle retries and have logic to ignore bad backends.

For more information, see [Use service discovery to connect Amazon ECS services with DNS names]

- *Amazon VPC Lattice*

  Amazon VPC Lattice is a managed application networking service that Amazon ECS customers use to observe, secure, and monitor applications built across AWS compute services, VPCs, and accounts without having to modify their code.

  VPC Lattice uses target groups, which are a collection of compute resources. These targets run your application or service and can be Amazon EC2 instances, IP addresses, Lambda functions, and Application Load Balancers. By associating their Amazon ECS services with a VPC Lattice target group, customers can now enable Amazon ECS tasks as IP targets in VPC Lattice. Amazon ECS automatically registers tasks to the VPC Lattice target group when tasks for the registered service are launched.

For more information, see Use Amazon VPC Lattice to connect, observe, and secure your Amazon ECS services.

Network mode compatibility table

The following table covers the compatibility between these options and the task network modes. In the table, "client" refers to the application that's making the connections from inside an Amazon ECS task.

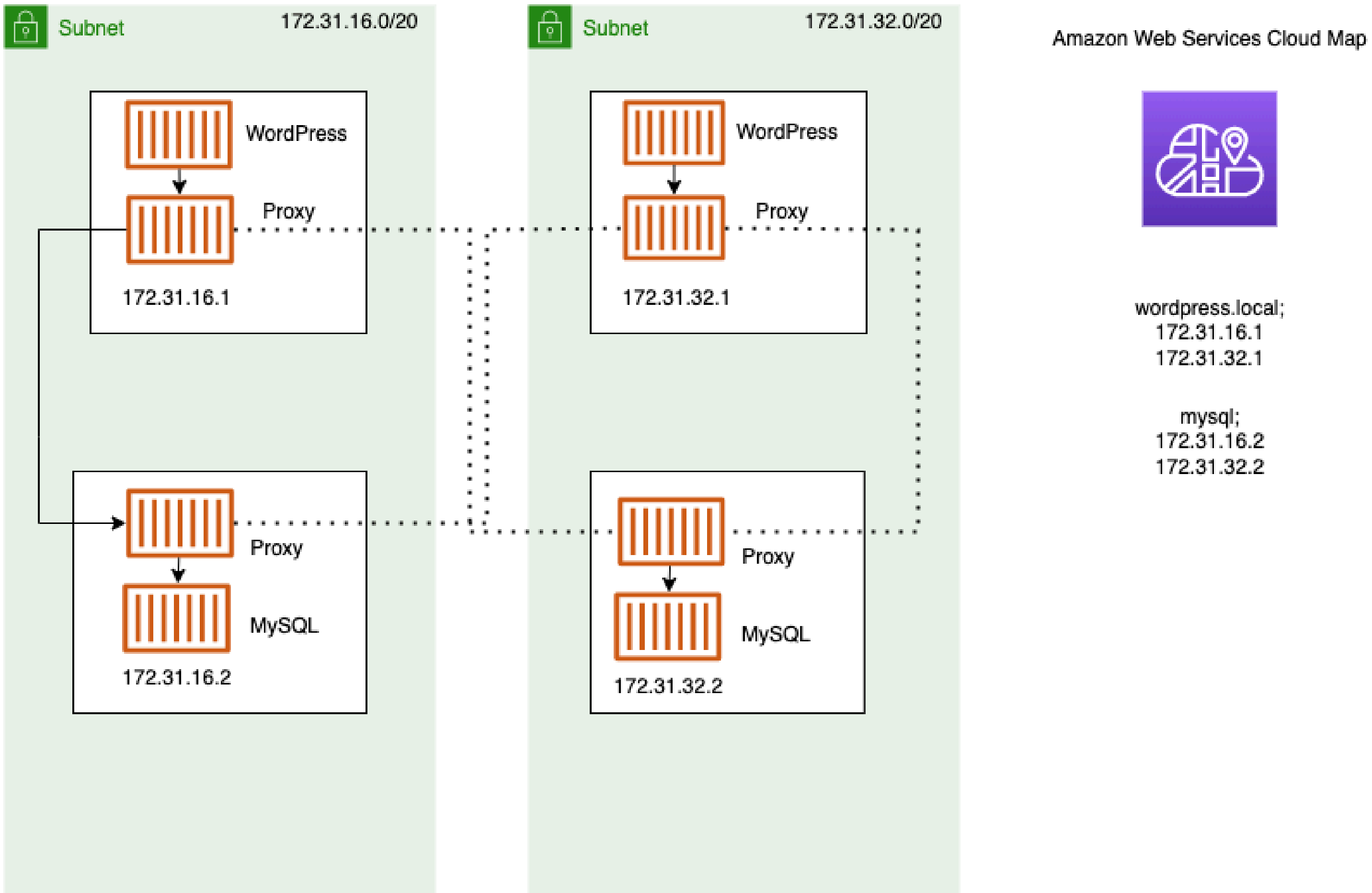
Interconnection Options	Bridged	`awsvpc`	Host
Service discovery	yes, but requires clients be aware of SRV records in DNS without `hostPort`.	yes	yes, but requires clients be aware of SRV records in DNS without `hostPort`.
Service Connect	yes	yes	no
VPC Lattice	yes	yes	yes

Use Service Connect to connect Amazon ECS services with short names

Amazon ECS Service Connect provides management of service-to-service communication as Amazon ECS configuration. It builds both service discovery and a service mesh in Amazon ECS. This provides the complete configuration inside each service that you manage by service deployments, a unified way to refer to your services within namespaces that doesn't depend on the VPC DNS configuration, and standardized metrics and logs to monitor all of your applications. Service Connect only interconnects services.

The following diagram shows an example Service Connect network with 2 subnets in the VPC and 2 services. A client service that runs WordPress with 1 task in each subnets. A server service that runs MySQL with 1 task in each subnet. Both services are highly available and resilient to task and Availability Zone issues because each service runs multiple tasks that are spread out over 2 subnets. The solid arrows show a connection from WordPress to MySQL. For example, a `mysql --host=mysql` CLI command that is run from inside the WordPress container in the task with the IP address `172.31.16.1`. The command uses the short name `mysql` on the default port for

MySQL. This name and port connects to the Service Connect proxy in the same task. The proxy in the WordPress task uses round-robin load balancing and any previous failure information in outlier detection to pick which MySQL task to connect to. As shown by the solid arrows in the diagram, the proxy connects to the second proxy in the MySQL task with the IP Address `172.31.16.2`. The second proxy connects to the local MySQL server in the same task. Both proxies report connection performance that is visible in graphs in the Amazon ECS and Amazon CloudWatch consoles so that you can get performance metrics from all kinds of applications in the same way.



The following terms are used with Service Connect.

port name

The Amazon ECS task definition configuration that assigns a name to a particular port mapping. This configuration is only used by Amazon ECS Service Connect.

client alias

The Amazon ECS service configuration that assigns the port number that is used in the endpoint. Additionally, the client alias can assign the DNS name of the endpoint, overriding the discovery name. If a discovery name isn't provided in the Amazon ECS service, the client alias name overrides the port name as the endpoint name. For endpoint examples, see the definition of *endpoint*. Multiple client aliases can be assigned to an Amazon ECS service. This configuration is only used by Amazon ECS Service Connect.

discovery name

The optional, intermediate name that you can create for a specified port from the task definition. This name is used to create a AWS Cloud Map service. If this name isn't provided, the port name from the task definition is used. Multiple discovery names can be assigned to a specific port an Amazon ECS service. This configuration is only used by Amazon ECS Service Connect.

AWS Cloud Map service names must be unique within a namespace. Because of this limitation, you can have only one Service Connect configuration without a discovery name for a particular task definition in each namespace.

endpoint

The URL to connect to an API or website. The URL contains the protocol, a DNS name, and the port. For more information about endpoints in general, see endpoint in the *AWS glossary* in the Amazon Web Services General Reference.

Service Connect creates endpoints that connect to Amazon ECS services and configures the tasks in Amazon ECS services to connect to the endpoints. The URL contains the protocol, a DNS name, and the port. You select the protocol and port name in the task definition, as the port must match the application that is inside the container image. In the service, you select each port by name and can assign the DNS name. If you don't specify a DNS name in the Amazon ECS service configuration, the port name from the task definition is used by default. For example, a Service Connect endpoint could be `http://blog:80`, `grpc://checkout:8080`, or `http://_db.production.internal:99`.

Service Connect service

The configuration of a single endpoint in an Amazon ECS service. This is a part of the Service Connect configuration, consisting of a single row in the **Service Connect and discovery name configuration** in the console, or one object in the `services` list in the JSON configuration of an Amazon ECS service. This configuration is only used by Amazon ECS Service Connect.

For more information, see ServiceConnectService in the Amazon Elastic Container Service API Reference.

namespace

The short name or full Amazon Resource Name (ARN) of the AWS Cloud Map namespace for use with Service Connect. The namespace must be in the same AWS Region as the Amazon ECS service and cluster. The type of namespace in AWS Cloud Map doesn't affect Service Connect.

Service Connect uses the AWS Cloud Map namespace as a logical grouping of Amazon ECS tasks that talk to one another. Each Amazon ECS service can belong to only one namespace. The services within a namespace can be spread across different Amazon ECS clusters within the same AWS Region in the same AWS account. You can freely organize services by any criteria.

client service

A service that runs a network client application. This service must have a namespace configured. Each task in the service can discover and connect to all of the endpoints in the namespace through a Service Connect proxy container.

If any of your containers in the task need to connect to an endpoint from a service in a namespace, choose a client service. If a frontend, reverse proxy, or load balancer application receives external traffic through other methods such as from Elastic Load Balancing, it could use this type of Service Connect configuration.

client-server service

An Amazon ECS service that runs a network or web service application. This service must have a namespace and at least one endpoint configured. Each task in the service is reachable by using the endpoints. The Service Connect proxy container listens on the endpoint name and port to direct traffic to the app containers in the task.

If any of the containers expose and listen on a port for network traffic, choose a client-server service. These applications don't need to connect to other client-server services in the same namespace, but the client configuration is needed. A backend, middleware, business tier, or most microservices can use this type of Service Connect configuration. If you want a frontend, reverse proxy, or load balancer application to receive traffic from other services configured with Service Connect in the same namespace, these services should use this type of Service Connect configuration.

The Service Connect feature creates a virtual network of related services. The same service configuration can be used across multiple different namespaces to run independent yet identical sets of applications. Service Connect defines the proxy container in the Amazon ECS service. This way, the same task definition can be used to run identical applications in different namespaces with different Service Connect configurations. Each task that the service makes runs a proxy container in the task.

Service Connect is suitable for connections between Amazon ECS services within the same namespace. For the following applications, you need to use an additional interconnection method to connect to an Amazon ECS service that is configured with Service Connect:

- Tasks that are configured in other namespaces
- Tasks that aren’t configured for Service Connect
- Other applications outside of Amazon ECS

These applications can connect through the Service Connect proxy but can’t resolve Service Connect endpoint names.

For these applications to resolve the IP addresses of Amazon ECS tasks, you need to use another interconnection method.

Topics

Pricing

- Amazon ECS Service Connect pricing depends on whether you use AWS Fargate or Amazon EC2 infrastructure to host your containerized workloads. When using Amazon ECS on AWS Outposts, the pricing follows the same model that's used when you use Amazon EC2 directly. For more information, see Amazon ECS Pricing.
- There is no additional charge for using Amazon ECS Service Connect.
- AWS Cloud Map usage is completely free when used by Service Connect.
- Customers pay for compute resources, used by Amazon ECS Service Connect, including vCPU and Memory. As Amazon ECS Service Connect agent runs inside a customer task, there is no additional charge for running it. The task resources are shared between the customer workload and the Amazon ECS Service Connect agent.

- When using Amazon ECS Service Connect traffic encryption functionality with AWS Private CA, customers pay for the private certificate authority they create and for each TLS certificate issued. For more details, see AWS Private Certificate Authority pricing. To estimate the monthly cost of TLS certificates, customers need to know the number of Amazon ECS services that have TLS enabled, multiply it by the certificate cost, and then multiply it by six. As Amazon ECS Service Connect automatically rotates TLS certificates every five days, there are six certificates issued per Amazon ECS service, per month, on average.

Amazon ECS Service Connect components

When you use Amazon ECS Service Connect, you configure each Amazon ECS service to run a server application that receives network requests (client-server service) or to run a client application that makes the requests (client service).

When you prepare to start using Service Connect, start with a client-server service. You can add a Service Connect configuration to a new service or an existing service. Amazon ECS creates a Service Connect endpoint in the namespace. Additionally, Amazon ECS creates a new deployment in the service to replace the tasks that are currently running.

Existing tasks and other applications can continue to connect to existing endpoints, and external applications. If a client-server service adds tasks by scaling out, new connections from clients will be balanced between all of the tasks. If a client-server service is updated, new connections from clients will be balanced between the tasks of the new version.

Existing tasks can't resolve and connect to the new endpoint. Only new tasks with a Service Connect configuration in the same namespace and that start running after this deployment can resolve and connect to this endpoint.

This means that the operator of the client application determines when the configuration of their app changes, even though the operator of the server application can change their configuration at any time. The list of endpoints in the namespace can change every time that any service in the namespace is deployed. Existing tasks and replacement tasks continue to behave the same as they did after the most recent deployment.

Consider the following examples.

First, assume that you are creating an application that is available to the public internet in a single AWS CloudFormation template and single AWS CloudFormation stack. The public discovery and reachability should be created last by AWS CloudFormation, including the frontend client service.

The services need to be created in this order to prevent an time period when the frontend client service is running and available the public, but a backend isn't. This eliminates error messages from being sent to the public during that time period. In AWS CloudFormation, you must use the `dependsOn` to indicate to AWS CloudFormation that multiple Amazon ECS services can't be made in parallel or simultaneously. You should add the `dependsOn` to the frontend client service for each backend client-server service that the client tasks connect to.

Second, assume that a frontend service exists without Service Connect configuration. The tasks are connecting to an existing backend service. Add a client-server Service Connect configuration to the backend service first, using the same name in the **DNS** or `clientAlias` that the frontend uses. This creates a new deployment, so all the deployment rollback detection or AWS Management Console, AWS CLI, AWS SDKs and other methods to roll back and revert the backend service to the previous deployment and configuration. If you are satisfied with the performance and behavior of the backend service, add a client or client-server Service Connect configuration to the frontend service. Only the tasks in the new deployment use the Service Connect proxy that is added to those new tasks. If you have issues with this configuration, you can roll back and revert to your previous configuration by using the deployment rollback detection or AWS Management Console, AWS CLI, AWS SDKs and other methods to roll back and revert the backend service to the previous deployment and configuration. If you use another service discovery system that is based on DNS instead of Service Connect, any frontend or client applications begin using new endpoints and changed endpoint configuration after the local DNS cache expires, commonly taking multiple hours.

Networking

By default, the Service Connect proxy listens on the `containerPort` from the task definition port mapping. Your security group rules must allow incoming (ingress) traffic to this port from the subnets where clients will run.

Even if you set a port number in the Service Connect service configuration, this doesn't change the port for the client-server service that the Service Connect proxy listens on. When you set this port number, Amazon ECS changes the port of the endpoint that the client services connect to, on the Service Connect proxy inside those tasks. The proxy in the client service connects to the proxy in the client-server service using the `containerPort`.

If you want to change the port that the Service Connect proxy listens on, change the `ingressPortOverride` in the Service Connect configuration of the client-server service. If you change this port number, you must allow inbound traffic on this port that is used by traffic to this service.

Traffic that your applications send to Amazon ECS services configured for Service Connect require that the Amazon VPC and subnets have route table rules and network ACL rules that allow the `containerPort` and `ingressPortOverride` port numbers that you are using.

You can use Service Connect to send traffic between VPCs. The same requirements for the route table rules, network ACLs, and security groups apply to both VPCs.

For example, two clusters create tasks in different VPCs. A service in each cluster is configured to use the same namespace. The applications in these two services can resolve every endpoint in the namespace without any VPC DNS configuration. However, the proxies can't connect unless the VPC peering, VPC, or subnet route tables, and VPC network ACLs allow the traffic on the `containerPort` and `ingressPortOverride` port numbers.

For tasks that use the `bridge` networking mode, you must create a security group with an inbound rule that allows traffic on the upper dynamic port range. Then, assign the security group to all the EC2 instances in the Service Connect cluster.

Service Connect proxy

If you create or update a service with Service Connect configuration, Amazon ECS adds a new container to each new task as it is started. This pattern of using a separate container is called a `sidecar`. This container isn't present in the task definition and you can't configure it. Amazon ECS manages the Container configuration in the service. This allows you to reuse the same task definitions between multiple services, namespaces, and tasks without Service Connect.

Proxy resources

- For task definitions, you must set the CPU and memory parameters.

  We recommend adding an additional 256 CPU units and at least 64 MiB of memory to your task CPU and memory for the Service Connect proxy container. On AWS Fargate, the lowest amount of memory that you can set is 512 MiB of memory. On Amazon EC2, task definition memory is required.
- For the service, you set the log configuration in the Service Connect configuration.
- If you expect tasks in this service to receive more than 500 requests per second at their peak load, we recommend adding 512 CPU units to your task CPU in this task definition for the Service Connect proxy container.
- If you expect to create more than 100 Service Connect services in the namespace or 2000 tasks in total across all Amazon ECS services within the namespace, we recommend adding 128 MiB

of memory to your task memory for the Service Connect proxy container. You should do this in every task definition that is used by all of the Amazon ECS services in the namespace.

Proxy configuration

Your applications connect to the proxy in the sidecar container in the same task as the application is in. Amazon ECS configures the task and containers so that applications only connect to the proxy when the application is connected to the endpoint names in the same namespace. All other traffic doesn't use the proxy. The other traffic includes IP addresses in the same VPC, AWS service endpoints, and external traffic.

Load balancing

Service Connect configures the proxy to use the round-robin strategy for load balancing between the tasks in a Service Connect endpoint. The local proxy that is in the task where the connection comes from, picks one of the tasks in the client-server service that provides the endpoint.

For example, consider a task that runs WordPress in a service that is configured as a *client service* in a namespace called *local*. There is another service with 2 tasks that run the MySQL database. This service is configured to provide an endpoint called `mysql` through Service Connect in the same namespace. In the WordPress task, the WordPress application connects to the database using the endpoint name. Connections to this name go to the proxy that runs in a sidecar container in the same task. Then, the proxy can connect to either of the MySQL tasks using the round-robin strategy.

Load balancing strategies: round-robin

Outlier detection

This feature uses data that the proxy has about prior failed connections to avoid sending new connections to the hosts that had the failed connections. Service Connect configures the outlier detection feature of the proxy to provide passive health checks.

Using the previous example, the proxy can connect to either of the MySQL tasks. If the proxy made multiple connections to a specific MySQL task, and 5 or more of the connections failed in the last 30 seconds, then the proxy avoids that MySQL task for 30 to 300 seconds.

Retries

Service Connect configures the proxy to retry connection that pass through the proxy and fail, and the second attempt avoids using the host from the previous connection. This ensures that each connection through Service Connect doesn't fail for one-off reasons.

Number of retries: 2

Timeout

Service Connect configures the proxy to wait a maximum time for your client-server applications to respond. The default timeout value is 15 seconds, but it can be updated.

Optional parameters:

idleTimeout - The amount of time in seconds a connection stays active while idle. A value of `0` disables `idleTimeout`.

The `idleTimeout` default for `HTTP`/`HTTP2`/`GRPC` is 5 minutes.

The `idleTimeout` default for `TCP` is 1 hour.

perRequestTimeout - The amount of time waiting for the upstream to respond with a complete response per request. A value of `0` turns off `perRequestTimeout`. This can only be set when the `appProtocol` for application container is `HTTP`/`HTTP2`/`GRPC`. The default is 15 seconds.

> **Note**
>
> If `idleTimeout` is set to a time that is less than `perRequestTimeout`, the connection will close when the `idleTimeout` is reached and not the `perRequestTimeout`.

Considerations

Consider the following when using Service Connect:

- Tasks that run in Fargate must use the Fargate Linux platform version `1.4.0` or higher to use Service Connect.
- The Amazon ECS agent version on the container instance must be `1.67.2` or higher.
- Container instances must run the Amazon ECS-optimized Amazon Linux 2023 AMI version `20230428` or later, or Amazon ECS-optimized Amazon Linux 2 AMI version `2.0.20221115` to

use Service Connect. These versions have the Service Connect agent in addition to the Amazon ECS container agent. For more information about the Service Connect agent, see Amazon ECS Service Connect Agent on GitHub.

- Container instances must have the `ecs:Poll` permission for the resource `arn:aws:ecs:region:0123456789012:task-set/cluster/*`. If you are using the `ecsInstanceRole`, you don't need to add additional permissions. The `AmazonEC2ContainerServiceforEC2Role` managed policy has the necessary permissions. For more information, see Amazon ECS container instance IAM role.
- Tasks that use the `bridge` network mode and use Service Connect don't support the `hostname` container definition parameter.
- Task definitions must set the task memory limit to use Service Connect. For more information, see Service Connect proxy.
- Task definitions that set container memory limits aren't supported.

  You can set container memory limits on your containers, but you must set the task memory limit to a number greater than the sum of the container memory limits. The additional CPU and memory in the task limits that aren't allocated in the container limits are used by the Service Connect proxy container and other containers that don't set container limits. For more information, see Service Connect proxy.
- You can configure Service Connect to use any AWS Cloud Map namespace in the same Region in the same AWS account.
- Each service can belong to only one namespace.
- Only the tasks that services create are supported.
- All endpoints must be unique within a namespace.
- All discovery names must be unique within a namespace.
- You must redeploy existing services before the applications can resolve new endpoints. New endpoints that are added to the namespace after the most recent deployment won't be added to the task configuration. For more information, see the section called “Service Connect components”.
- Service Connect doesn't delete namespaces when clusters are deleted. You must delete namespaces in AWS Cloud Map.
- Application Load Balancer traffic defaults to routing through the Service Connect agent in `awsvpc` network mode. If you want non-service traffic to bypass the Service Connect agent, use the `ingressPortOverride` parameter in your Service Connect service configuration.

- Service Connect with TLS does not support Bridge networking mode. Only `awsvpc` networking mode is supported.

Service Connect doesn't support the following:

- Windows containers
- HTTP 1.0
- Standalone tasks
- Services that use the blue/green powered by CodeDeploy and external deployment types
- `External` container instance for Amazon ECS Anywhere aren't supported with Service Connect.
- PPv2
- FIPS mode

Amazon ECS Service Connect configuration overview

When you use Service Connect, there are parameters you need to configure in your resources.

The following table describes the configuration parameters for the Amazon ECS resources.

Parameter location	App type	Description	Required
Task definition	Client	There are no changes available for Service Connect in client task definitions.	N/A
Task definition	Client-server	Servers must add `name` fields to ports in the `portMappings` of containers. For more information, see [`portMappings`](#)	Yes
Task definition	Client-server	Servers can optionally provide an application protocol (for example, HTTP) to receive protocol-specific metrics for their server applications (for example, `HTTP 5xx`).	No

Parameter location	App type	Description	Required
Service definition	Client	Client services must add a `serviceConnectConfiguration` to configure the namespace to join. This namespace must contain all of the server services that this service needs to discover. For more information, see [`serviceConnectConfiguration`]().	Yes
Service definition	Client-server	Server services must add a `serviceConnectConfiguration` to configure the DNS names, port numbers, and namespace that the service is available from. For more information, see [`serviceConnectConfiguration`]().	Yes
Cluster	Client	Clusters can add a default Service Connect namespace. New services in the cluster inherit the namespace when Service Connect is configured in a service.	No
Cluster	Client-server	There are no changes available for Service Connect in clusters that apply to server services. Server task definitions and services must set the respective configuration.	N/A

Overview of steps to configure Service Connect

The following steps provide an overview of how to configure Service Connect.

> **Important**
>
> - Service Connect creates AWS Cloud Map services in your account. Modifying these AWS Cloud Map resources by manually registering/deregistering instances, changing instance attributes, or deleting a service may lead to unexpected behavior for your application traffic or subsequent deployments.
> - Service Connect doesn't support links in the task definition.

1. Add port names to the port mappings in your task definitions. Additionally, you can identify the layer 7 protocol of the application, to get additional metrics.
2. Create a cluster with a AWS Cloud Map namespace or create the namespace separately. For simple organization, create a cluster with the name that you want for the namespace and specify the identical name for the namespace. In this case, Amazon ECS creates a new HTTP namespace with the necessary configuration. Service Connect doesn't use or create DNS hosted zones in Amazon Route 53.
3. Configure services to create Service Connect endpoints within the namespace.
4. Deploy services to create the endpoints. Amazon ECS adds a Service Connect proxy container to each task, and creates the Service Connect endpoints in AWS Cloud Map. This container isn't configured in the task definition, and the task definition can be reused without modification to create multiple services in the same namespace or in multiple namespaces.
5. Deploy client apps as services to connect to the endpoints. Amazon ECS connects them to the Service Connect endpoints through the Service Connect proxy in each task.

   Applications only use the proxy to connect to Service Connect endpoints. There is no additional configuration to use the proxy. The proxy performs round-robin load balancing, outlier detection, and retries. For more information about the proxy, see Service Connect proxy.
6. Monitor traffic through the Service Connect proxy in Amazon CloudWatch.

Cluster configuration

You can set a default namespace for Service Connect when you create or update the cluster. If you specify a namespace name that doesn't exist in the same AWS Region and account, a new HTTP namespace is created.

If you create a cluster and specify a default Service Connect namespace, the cluster waits in the `PROVISIONING` status while Amazon ECS creates the namespace. You can see an `attachment` in the status of the cluster that shows the status of the namespace. Attachments aren't displayed by default in the AWS CLI, you must add `--include ATTACHMENTS` to see them.

Service configuration

Service Connect is designed to require the minimum configuration. You need to set a name for each port mapping that you would like to use with Service Connect in the task definition. In the service, you need to turn on Service Connect and select a namespace to make a client service. To make a client-server service, you need to add a single Service Connect service configuration that matches the name of one of the port mappings. Amazon ECS reuses the port number and port name from the task definition to define the Service Connect service and endpoint. To override those values, you can use the other parameters **Discovery**, **DNS**, and **Port** in the console, or `discoveryName` and `clientAliases`, respectively in the Amazon ECS API.

Encrypt Amazon ECS Service Connect traffic

Amazon ECS Service Connect supports automatic traffic encryption with Transport Layer Security (TLS) certificates for Amazon ECS services. When you point your Amazon ECS services toward an AWS Private Certificate Authority (AWS Private CA), Amazon ECS automatically provisions TLS certificates to encrypt traffic between your Amazon ECS Service Connect services. Amazon ECS generates, rotates, and distributes TLS certificates used for traffic encryption.

Automatic traffic encryption with Service Connect uses industry-leading encryption capabilities to secure your inter-service communication that helps you meet your security requirements. It supports AWS Private Certificate Authority TLS certificates with `256-bit ECDSA` and `2048-bit RSA` encryption. You also have full control over private certificates and signing keys to help you meet compliance requirements. By default, TLS 1.3 is supported, but TLS 1.0 - 1.2 are not supported. Service Connect supports TLS 1.3 with the following ciphers:

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_CHACHA20_POLY1305_SHA256`

> **Note**
>
> In order to use TLS 1.3, you must enable it on the listener on the target.

Only inbound and outbound traffic passing though the Amazon ECS agent is encrypted.

Service Connect and Application Load Balancer health checks

You can use Service Connect with Application Load Balancer health checks and TLS 1.3 encryption.

Application Load Balancer configuration

Configure the Application Load Balancer with the following settings:

- Configure a TLS listener with a TLS 1.3 security policy (such as `ELBSecurityPolicy-TLS13-1-2-2021-06`). For more information, see Security policies for your Application Load Balancer.
- Create a target group with the following settings:
  - Set the protocol to HTTPS
  - Attach the target group to the TLS listener
  - Configure the health check port to match your Service Connect service's container port

Service Connect configuration

Configure a service with the following settings:

- Configure the service to use `awsvpc` network mode, as `bridge` network mode is not supported.
- Enable Service Connect for the service.
- Set up the load balancer configuration with the following settings:
  - Specify the target group you configured for your Application Load Balancer
  - Set the container port to match the Service Connect TLS service's container port
- Avoid setting `ingressPortOverride` for the service. For more information, see ServiceConnectService in the *Amazon Elastic Container Service API Reference*.

Considerations

Consider the following when using Application Load Balancer, TLS and Service Connect:

- Use `awsvpc` network mode instead of `bridge` network mode for HTTPS health checks when using Service Connect with TLS encryption. HTTP health checks will continue to work with `bridge` mode.
- Configure the target group health check port to match the Service Connect service's container port, not the default HTTPS port (443).

AWS Private Certificate Authority certificates and Service Connect

You need to have the infrastructure IAM role. For more information about this role, see Amazon ECS infrastructure IAM role.

AWS Private Certificate Authority modes for Service Connect

AWS Private Certificate Authority can run in two modes: general purpose and short-lived.

- General purpose - Certificates that can be configured with any expiration date.
- Short-lived - Certificates with a maximum validity of seven days.

While Amazon ECS supports both modes, we recommend using short-lived certificates. By default, certificates rotate every five days, and running in short-lived mode offers significant cost savings over general purpose.

Service Connect doesn't support certificate revocation and instead leverages short-lived certificates with frequent certificate rotation. You have the authority to modify the rotation frequency, disable, or delete the secrets using managed rotation in Secrets Manager, but doing so can come with the following possible consequences.

- Shorter Rotation Frequency - A shorter rotation frequency incurs higher costs due to AWS Private CA, AWS KMS and Secrets Manager, and Auto Scaling experiencing an increased workload for rotation.
- Longer Rotation Frequency - Your applications’ communications fail if the rotation frequency exceeds **seven** days.
- Deletion of Secret - Deleting the secret results in rotation failure and impacts customer application communications.

In the event of your secret rotation failing, a `RotationFailed` event is published in AWS CloudTrail. You can also set up a CloudWatch Alarm for `RotationFailed`.

> **Important**
>
> Don't add replica Regions to secrets. Doing so prevents Amazon ECS from deleting the secret, because Amazon ECS doesn't have permission to remove Regions from replication. If you already added the replication, run the following command.
>
> ```
> aws secretsmanager remove-regions-from-replication \
>  --secret-id SecretId \
>  --remove-replica-regions region-name
> ```

Subordinate Certificate Authorities

You can bring any AWS Private CA, root or subordinate, to Service Connect TLS to issue end-entity certificates for the services. The provided issuer is treated as the signer and root of trust everywhere. You can issue end-entity certificates for different parts of your application from different subordinate CAs. When using the AWS CLI, provide the Amazon Resource Name (ARN) of the CA to establish the trust chain.

On-premises Certificate Authorities

To use your on-premises CA, you create and configure a subordinate CA in AWS Private Certificate Authority. This ensures all TLS certificates issued for your Amazon ECS workloads share the trust chain with the workloads you run on premises and are able to securely connect.

> **Important**
>
> Add the **required** tag `AmazonECSManaged : true` in your AWS Private CA.

Infrastructure as code

When using Service Connect TLS with Infrastructure as Code (IaC) tools, it's important to configure your dependencies correctly to avoid issues, such as services stuck in draining. Your AWS KMS key, if provided, IAM role, and AWS Private CA dependencies should be deleted after your Amazon ECS service.

Service Connect and Secrets Manager

When using Amazon ECS Service Connect with TLS encryption, the service interacts with Secrets Manager in the following ways:

Service Connect utilizes the infrastructure role provided to create secrets within Secrets Manager. These secrets are used to store the associated private keys for your TLS certificates for encrypting traffic between your Service Connect services.

> **Warning**
>
> The automatic creation and management of these secrets by Service Connect streamlines the process of implementing TLS encryption for your services. However, it's important to be aware of potential security implications. Other IAM roles that have read access to Secrets Manager may be able to access these automatically created secrets. This could expose sensitive cryptographic material to unauthorized parties, if access controls are not properly configured.
> To mitigate this risk, follow these best practices:
>
> - Carefully manage and restrict access to Secrets Manager, especially for secrets created by Service Connect.
> - Regularly audit IAM roles and their permissions to ensure the principle of least privilege is maintained.

When granting read access to Secrets Manager, consider excluding the TLS private keys created by Service Connect. You can do this by using a condition in your IAM policies to exclude secrets with ARNs that match the pattern:

```
"arn:aws:secretsmanager:::secret:ecs-sc!"
```

An example IAM policy that denies the `GetSecretValue` action to all secrets with the `ecs-sc!` prefix:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Deny",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:*:*:secret:ecs-sc!*"
        }
    ]
}
```

> **Note**
>
> This is a general example and may need to be adjusted based on your specific use case and AWS account configuration. Always test your IAM policies thoroughly to ensure they provide the intended access while maintaining security.

By understanding how Service Connect interacts with Secrets Manager, you can better manage the security of your Amazon ECS services while leveraging the benefits of automatic TLS encryption.

Service Connect and AWS Key Management Service

You can use AWS Key Management Service to encrypt and decrypt your Service Connect resources. AWS KMS is a service managed by AWS where you can make and manage cryptographic keys that protect your data.

When using AWS KMS with Service Connect, you can either choose to use an AWS owned key that AWS manages for you, or you can choose an existing AWS KMS key. You can also create a new AWS KMS key to use.

Providing your own encryption key

You can provide your own key materials, or you can use an external key store through AWS Key Management Service Import your own key into AWS KMS, and then specify the Amazon Resource Name (ARN) of that key in Amazon ECS Service Connect.

The following is an example AWS KMS policy. Replace the *user input* values with your own.

JSON

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "id",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/role-name"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyPair"
      ],
      "Resource": "*"
    }
  ]
}
```

For more information about key policies, see Creating a key policy in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> Service Connect supports only symmetric encryption AWS KMS keys. You can't use any other type of AWS KMS key to encrypt your Service Connect resources. For help determining whether a AWS KMS key is a symmetric encryption key, see Identify asymmetric KMS keys.

For more information on AWS Key Management Service symmetric encryption keys, see Symmetric encryption AWS KMS keys in the *AWS Key Management Service Developer Guide*.

Enabling TLS for Amazon ECS Service Connect

You enable traffic encryption when you create or update a Service Connect service.

To enable traffic encryption for a service in an existing namespace using the AWS Management Console

1. You need to have the infrastructure IAM role. For more information about this role, see Amazon ECS infrastructure IAM role.

2. Open the console at https://console.aws.amazon.com/ecs/v2.
3. In the navigation pane, choose **Namespaces**.
4. Choose the **Namespace** with the **Service** you'd like to enable traffic encryption for.
5. Choose the **Service** you'd like to enable traffic encryption for.
6. Choose **Update Service** in the top right corner and scroll down to the Service Connect section.
7. Choose **Turn on traffic encryption** under your service information to enable TLS.
8. For **Service Connect TLS role**, choose an existing infrastructure IAM role or create a new one.
9. For **Signer certificate authority**, choose an existing certificate authority or create a new one.

   For more information, see see AWS Private Certificate Authority certificates and Service Connect.
10. For **Choose an AWS KMS key**, choose an AWS owned and managed key or you can choose a different key. You can also choose to create a new one.

For an example of using the AWS CLI to configure TLS for your service, Configuring Amazon ECS Service Connect with the AWS CLI.

Verifying TLS is enabled for Amazon ECS Service Connect

Service Connect initiates TLS at the Service Connect agent and terminates it at the destination agent. As a result, the application code never sees TLS interactions. Use the following steps to verify that TLS is enabled.

1. Include the `openssl` CLI in your application image.
2. Enable ECS Exec on your services to connect to your tasks via SSM. Alternately, you can launch an Amazon EC2 instance in the same Amazon VPC as the service.
3. Retrieve the IP and port of a task from a service that you want to verify. You can retrieve the task IP address in the AWS Cloud Map console. The information is on the service details page for the namespace.
4. Log on to any of your tasks using `execute-command` like in the following example. Alternately, log on to the Amazon EC2 instance created in **Step 2**.

```
$ aws ecs execute-command --cluster cluster-name \
    --task task-id  \
    --container container-name \
    --interactive \
```

```
    --command "/bin/sh"
```

> **Note**
>
> Calling the DNS name directly does not reveal the certificate.

5. In the connected shell, use the `openssl` CLI to verify and view the certificate attached to the task.

   Example:

   ```
   openssl s_client -connect 10.0.147.43:6379 < /dev/null 2> /dev/null \
   | openssl x509 -noout -text
   ```

   Example response:

   ```
   Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number:
               <serial-number>
           Signature Algorithm: ecdsa-with-SHA256
           Issuer: <issuer>
           Validity
               Not Before: Jan 23 21:38:12 2024 GMT
               Not After : Jan 30 22:38:12 2024 GMT
           Subject: <subject>
           Subject Public Key Info:
               Public Key Algorithm: id-ecPublicKey
                   Public-Key: (256 bit)
                   pub:
                       <pub>
                   ASN1 OID: prime256v1
                   NIST CURVE: P-256
           X509v3 extensions:
               X509v3 Subject Alternative Name:
                   DNS:redis.yelb-cftc
               X509v3 Basic Constraints:
                   CA:FALSE
               X509v3 Authority Key Identifier:
                   keyid:<key-id>
   ```

```
            X509v3 Subject Key Identifier:
                1D:<id>
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
    Signature Algorithm: ecdsa-with-SHA256
        <hash>
```

Configuring Amazon ECS Service Connect with the AWS CLI

You can create an Amazon ECS service for a Fargate task that uses Service Connect with the AWS CLI.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

Prerequisites

The following are Service Connect prerequisites:

- Verify that the latest version of the AWS CLI is installed and configured. For more information, see Installing or updating to the latest version of the AWS CLI.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- You have a VPC, subnet, route table, and security group created to use. For more information, see the section called “Create a virtual private cloud”.
- You have a task execution role with the name `ecsTaskExecutionRole` and the `AmazonECSTaskExecutionRolePolicy` managed policy is attached to the role. This role allows Fargate to write the NGINX application logs and Service Connect proxy logs to Amazon CloudWatch Logs. For more information, see Creating the task execution role.

Step 1: Create the cluster

Use the following steps to create your Amazon ECS cluster and namespace.

To create the Amazon ECS cluster and AWS Cloud Map namespace

1. Create an Amazon ECS cluster named `tutorial` to use. The parameter `--service-connect-defaults` sets the default namespace of the cluster. In the example output, a AWS Cloud Map namespace of the name `service-connect` doesn't exist in this account and AWS Region, so the namespace is created by Amazon ECS. The namespace is made in AWS Cloud Map in the account, and is visible with all of the other namespaces, so use a name that indicates the purpose.

```
aws ecs create-cluster --cluster-name tutorial --service-connect-defaults
 namespace=service-connect
```

Output:

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
        "clusterName": "tutorial",
        "serviceConnectDefaults": {
            "namespace": "arn:aws:servicediscovery:us-
west-2:123456789012:namespace/ns-EXAMPLE"
        },
        "status": "PROVISIONING",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "capacityProviders": [],
        "defaultCapacityProviderStrategy": [],
        "attachments": [
```

```
            {
                "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                "type": "sc",
                "status": "ATTACHING",
                "details": []
            }
        ],
        "attachmentsStatus": "UPDATE_IN_PROGRESS"
    }
}
}
```

2. Verify that the cluster is created:

```
aws ecs describe-clusters --clusters tutorial
```

Output:


```
{
    "clusters": [
        {
            "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
            "clusterName": "tutorial",
            "serviceConnectDefaults": {
                "namespace": "arn:aws:servicediscovery:us-
west-2:123456789012:namespace/ns-EXAMPLE"
            },
            "status": "ACTIVE",
            "registeredContainerInstancesCount": 0,
            "runningTasksCount": 0,
            "pendingTasksCount": 0,
            "activeServicesCount": 0,
            "statistics": [],
            "tags": [],
            "settings": [],
            "capacityProviders": [],
            "defaultCapacityProviderStrategy": []
        }
    ],
    "failures": []
}
```

3. (Optional) Verify that the namespace is created in AWS Cloud Map. You can use the AWS Management Console or the normal AWS CLI configuration as this is created in AWS Cloud Map.

   For example, use the AWS CLI:

   ```
   aws servicediscovery get-namespace --id ns-EXAMPLE
   ```

   Output:

   ```
   {
       "Namespace": {
           "Id": "ns-EXAMPLE",
           "Arn": "arn:aws:servicediscovery:us-west-2:123456789012:namespace/ns-
   EXAMPLE",
           "Name": "service-connect",
           "Type": "HTTP",
           "Properties": {
               "DnsProperties": {
                   "SOA": {}
               },
               "HttpProperties": {
                   "HttpName": "service-connect"
               }
           },
           "CreateDate": 1661749852.422,
           "CreatorRequestId": "service-connect"
       }
   }
   ```

Step 2: Create the service for the server

The Service Connect feature is intended for interconnecting multiple applications on Amazon ECS. At least one of those applications needs to provide a web service to connect to. In this step, you create:

- The task definition that uses the unmodified official NGINX container image and includes Service Connect configuration.

- The Amazon ECS service definition that configures Service Connect to provide service discovery and service mesh proxy for traffic to this service. The configuration reuses the default namespace from the cluster configuration to reduce the amount of service configuration that you make for each service.
- The Amazon ECS service. It runs one task using the task definition, and inserts an additional container for the Service Connect proxy. The proxy listens on the port from the container port mapping of the task definition. In a client application running in Amazon ECS, the proxy in the client task listens for outbound connections to the task definition port name, service discovery name or service client alias name, and the port number from the client alias.

To create the web service with Service Connect

1. Register a task definition that's compatible with Fargate and uses the `awsvpc` network mode. Follow these steps:

   a. Create a file that's named `service-connect-nginx.json` with the contents of the following task definition.

      This task definition configures Service Connect by adding `name` and `appProtocol` parameters to the port mapping. The port name makes this port more identifiable in the service configuration when multiple ports are used. The port name is also used by default as the discoverable name for use by other applications in the namespace.

      The task definition contains the task IAM role because the service has ECS Exec enabled.

      Important

      This task definition uses a `logConfiguration` to send the nginx output from `stdout` and `stderr` to Amazon CloudWatch Logs. This task execution role doesn't have the extra permissions required to make the CloudWatch Logs log group. Create the log group in CloudWatch Logs using the AWS Management Console or AWS CLI. If you don't want to send the nginx logs to CloudWatch Logs you can remove the `logConfiguration`.
      Replace the AWS account id in the task execution role with your AWS account id.

```
{
    "family": "service-connect-nginx",
    "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
    "taskRoleArn": "arn:aws:iam::123456789012:role/ecsTaskRole",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
        "name": "webserver",
        "image": "public.ecr.aws/docker/library/nginx:latest",
        "cpu": 100,
        "portMappings": [
            {
                "name": "nginx",
                "containerPort": 80,
                "protocol": "tcp",
                "appProtocol": "http"
            }
        ],
        "essential": true,
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-nginx",
                "awslogs-region": "region",
                "awslogs-stream-prefix": "nginx"
            }
        }
        }
    ],
    "cpu": "256",
    "memory": "512"
}
```

b. Register the task definition using the `service-connect-nginx.json` file:

```
aws ecs register-task-definition --cli-input-json file://service-connect-
nginx.json
```

2. Create a service:

   a. Create a file that's named `service-connect-nginx-service.json` with the contents of the Amazon ECS service that you're creating. This example uses the task definition that was created in the previous step. An `awsvpcConfiguration` is required because the example task definition uses the `awsvpc` network mode.

When you create the ECS service, specify the Fargate launch type, and the `LATEST` platform version that supports Service Connect. The `securityGroups` and `subnets` must belong to a VPC that has the requirements for using Amazon ECS. You can obtain the security group and subnet IDs from the Amazon VPC Console.

This service configures Service Connect by adding the `serviceConnectConfiguration` parameter. The namespace is not required because the cluster has a default namespace configured. Client applications running in ECS in the namespace connect to this service by using the `portName` and the port in the `clientAliases`. For example, this service is reachable using `http://nginx:80/`, as nginx provides a welcome page in the root location `/`. External applications that are not running in Amazon ECS or are not in the same namespace can reach this application through the Service Connect proxy by using the IP address of the task and the port number from the task definition. For your `tls` configuration, add the certificate `arn` for your `awsPcaAuthorityArn`, your `kmsKey`, and `roleArn` of your IAM role.

This service uses a `logConfiguration` to send the service connect proxy output from `stdout` and `stderr` to Amazon CloudWatch Logs. This task execution role doesn't have the extra permissions required to make the CloudWatch Logs log group. Create the log group in CloudWatch Logs using the AWS Management Console or AWS CLI. We recommend that you create this log group and store the proxy logs in CloudWatch Logs. If you don't want to send the proxy logs to CloudWatch Logs you can remove the `logConfiguration`.

```
{
    "cluster": "tutorial",
    "deploymentConfiguration": {
        "maximumPercent": 200,
        "minimumHealthyPercent": 0
    },
    "deploymentController": {
        "type": "ECS"
    },
    "desiredCount": 1,
    "enableECSManagedTags": true,
```

```
    "enableExecuteCommand": true,
    "launchType": "FARGATE",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "assignPublicIp": "ENABLED",
            "securityGroups": [
                "sg-EXAMPLE"
            ],
            "subnets": [
                "subnet-EXAMPLE",
                "subnet-EXAMPLE",
                "subnet-EXAMPLE"
            ]
           }
    },
    "platformVersion": "LATEST",
    "propagateTags": "SERVICE",
    "serviceName": "service-connect-nginx-service",
    "serviceConnectConfiguration": {
        "enabled": true,
        "services": [
            {
                "portName": "nginx",
                "clientAliases": [
                    {
                        "port": 80
                    }
                ],
                "tls": {
                   "issuerCertificateAuthority": {
                      "awsPcaAuthorityArn": "certificateArn"
                   },
                   "kmsKey": "kmsKey",
                   "roleArn": "iamRoleArn"
                }
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "/ecs/service-connect-proxy",
                "awslogs-region": "region",
                "awslogs-stream-prefix": "service-connect-proxy"
            }
```

```
        }
    },
    "taskDefinition": "service-connect-nginx"
}
```

b. Create a service using the `service-connect-nginx-service.json` file:

```
aws ecs create-service --cluster tutorial --cli-input-json file://service-connect-nginx-service.json
```

Output:


```
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-west-2:123456789012:service/tutorial/service-connect-nginx-service",
        "serviceName": "service-connect-nginx-service",
        "clusterArn": "arn:aws:ecs:us-west-2:123456789012:cluster/tutorial",
        "loadBalancers": [],
        "serviceRegistries": [],
        "status": "ACTIVE",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "FARGATE",
        "platformVersion": "LATEST",
        "platformFamily": "Linux",
        "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-definition/service-connect-nginx:1",
        "deploymentConfiguration": {
            "deploymentCircuitBreaker": {
                "enable": false,
                "rollback": false
            },
            "maximumPercent": 200,
            "minimumHealthyPercent": 0
        },
        "deployments": [
            {
                "id": "ecs-svc/3763308422771520962",
                "status": "PRIMARY",
```

```
                "taskDefinition": "arn:aws:ecs:us-west-2:123456789012:task-
definition/service-connect-nginx:1",
                "desiredCount": 1,
                "pendingCount": 0,
                "runningCount": 0,
                "failedTasks": 0,
                "createdAt": 1661210032.602,
                "updatedAt": 1661210032.602,
                "launchType": "FARGATE",
                "platformVersion": "1.4.0",
                "platformFamily": "Linux",
                "networkConfiguration": {
                    "awsvpcConfiguration": {
                        "assignPublicIp": "ENABLED",
                        "securityGroups": [
                            "sg-EXAMPLE"
                        ],
                        "subnets": [
                            "subnet-EXAMPLEf",
                            "subnet-EXAMPLE",
                            "subnet-EXAMPLE"
                        ]
                    }
                },
                "rolloutState": "IN_PROGRESS",
                "rolloutStateReason": "ECS deployment ecs-
svc/3763308422771520962 in progress.",
                "failedLaunchTaskCount": 0,
                "replacedTaskCount": 0,
                "serviceConnectConfiguration": {
                    "enabled": true,
                    "namespace": "service-connect",
                    "services": [
                        {
                            "portName": "nginx",
                            "clientAliases": [
                                {
                                    "port": 80
                                }
                            ]
                        }
                    ],
                    "logConfiguration": {
                        "logDriver": "awslogs",
```

```
                        "options": {
                            "awslogs-group": "/ecs/service-connect-proxy",
                            "awslogs-region": "us-west-2",
                            "awslogs-stream-prefix": "service-connect-proxy"
                        },
                        "secretOptions": []
                    }
                },
                "serviceConnectResources": [
                    {
                        "discoveryName": "nginx",
                        "discoveryArn": "arn:aws:servicediscovery:us-
west-2:123456789012:service/srv-EXAMPLE"
                    }
                ]
            }
        ],
        "roleArn": "arn:aws:iam::123456789012:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
        "version": 0,
        "events": [],
        "createdAt": 1661210032.602,
        "placementConstraints": [],
        "placementStrategy": [],
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "assignPublicIp": "ENABLED",
                "securityGroups": [
                    "sg-EXAMPLE"
                ],
                "subnets": [
                    "subnet-EXAMPLE",
                    "subnet-EXAMPLE",
                    "subnet-EXAMPLE"
                ]
            }
        },
        "schedulingStrategy": "REPLICA",
        "enableECSManagedTags": true,
        "propagateTags": "SERVICE",
        "enableExecuteCommand": true
    }
}
```

The `serviceConnectConfiguration` that you provided appears inside the first *deployment* of the output. As you make changes to the ECS service in ways that need to make changes to tasks, a new deployment is created by Amazon ECS.

Step 3: Verify that you can connect

To verify that Service Connect is configured and working, follow these steps to connect to the web service from an external application. Then, see the additional metrics in CloudWatch the Service Connect proxy creates.

To connect to the web service from an external application

- Connect to the task IP address and container port using the task IP address

  Use the AWS CLI to get the task ID, using the `aws ecs list-tasks --cluster tutorial`.

  If your subnets and security group permit traffic from the public internet on the port from the task definition, you can connect to the public IP from your computer. The public IP isn't available from `describe-tasks` however, so the steps involve going to the Amazon EC2 AWS Management Console or AWS CLI to get the details of the elastic network interface.

  In this example, an Amazon EC2 instance in the same VPC uses the private IP of the task. The application is nginx, but the `server: envoy` header shows that the Service Connect proxy is used. The Service Connect proxy is listening on the container port from the task definition.

```
$ curl -v 10.0.19.50:80/
*   Trying 10.0.19.50:80...
* Connected to 10.0.19.50 (10.0.19.50) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.0.19.50
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< server: envoy
< date: Tue, 23 Aug 2022 03:53:06 GMT
< content-type: text/html
```

```
< content-length: 612
< last-modified: Tue, 16 Apr 2019 13:08:19 GMT
< etag: "5cb5d3c3-264"
< accept-ranges: bytes
< x-envoy-upstream-service-time: 0
<
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

To view Service Connect metrics

The Service Connect proxy creates application (HTTP, HTTP2, gRPC, or TCP connection) metrics in CloudWatch metrics. When you use the CloudWatch console, see the additional metric dimensions of **DiscoveryName**, (**DiscoveryName, ServiceName, ClusterName**), **TargetDiscoveryName**, and (**TargetDiscoveryName, ServiceName, ClusterName**) under the Amazon ECS namespace. For more information about these metrics and the dimensions, see [View Available Metrics](#) in the Amazon CloudWatch Logs User Guide.

Use service discovery to connect Amazon ECS services with DNS names

Your Amazon ECS service can optionally be configured to use Amazon ECS service discovery. Service discovery uses AWS Cloud Map API actions to manage HTTP and DNS namespaces for your Amazon ECS services. For more information, see What Is AWS Cloud Map in the *AWS Cloud Map Developer Guide*.

Service discovery is available in the following AWS Regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Canada (Central)	ca-central-1

Region Name	Region
Canada West (Calgary)	ca-west-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Stockholm)	eu-north-1
Israel (Tel Aviv)	il-central-1
Europe (Spain)	eu-south-2
Middle East (UAE)	me-central-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

Service Discovery concepts

Service discovery consists of the following components:

- **Service discovery namespace**: A logical group of service discovery services that share the same domain name, such as `example.com`, which is where you want to route traffic. You can create a namespace with a call to the `aws servicediscovery create-private-dns-namespace` command or in the Amazon ECS console. You can use the `aws servicediscovery list-namespaces` command to view the summary information about the namespaces that were created by the current account. For more information about the service discovery commands, see [`create-private-dns-namespace`]() and [`list-namespaces`]() in the *AWS Cloud Map (service discovery) AWS CLI Reference Guide*.
- **Service discovery service**: Exists within the service discovery namespace and consists of the service name and DNS configuration for the namespace. It provides the following core component:
  - **Service registry**: Allows you to look up a service via DNS or AWS Cloud Map API actions and get back one or more available endpoints that can be used to connect to the service.
- **Service discovery instance**: Exists within the service discovery service and consists of the attributes associated with each Amazon ECS service in the service directory.
  - **Instance attributes**: The following metadata is added as custom attributes for each Amazon ECS service that is configured to use service discovery:
    - **`AWS_INSTANCE_IPV4`** – For an A record, the IPv4 address that Route 53 returns in response to DNS queries and AWS Cloud Map returns when discovering instance details, for example, `192.0.2.44`.
    - **`AWS_INSTANCE_PORT`** – The port value associated with the service discovery service.
    - **`AVAILABILITY_ZONE`** – The Availability Zone into which the task was launched. For tasks using the EC2 launch type, this is the Availability Zone in which the container instance exists. For tasks using the Fargate launch type, this is the Availability Zone in which the elastic network interface exists.
    - **`REGION`** – The Region in which the task exists.
    - **`ECS_SERVICE_NAME`** – The name of the Amazon ECS service to which the task belongs.
    - **`ECS_CLUSTER_NAME`** – The name of the Amazon ECS cluster to which the task belongs.
    - **`EC2_INSTANCE_ID`** – The ID of the container instance the task was placed on. This custom attribute is not added if the task is using the Fargate launch type.
    - **`ECS_TASK_DEFINITION_FAMILY`** – The task definition family that the task is using.
    - **`ECS_TASK_SET_EXTERNAL_ID`** – If a task set is created for an external deployment and is associated with a service discovery registry, then the `ECS_TASK_SET_EXTERNAL_ID` attribute will contain the external ID of the task set.

- **Amazon ECS health checks**: Amazon ECS performs periodic container-level health checks. If an endpoint does not pass the health check, it is removed from DNS routing and marked as unhealthy.

Service discovery considerations

The following should be considered when using service discovery:

- Service discovery is supported for tasks on Fargate that use platform version 1.1.0 or later. For more information, see Fargate platform versions for Amazon ECS.
- Services configured to use service discovery have a limit of 1,000 tasks per service. This is due to a Route 53 service quota.
- The Create Service workflow in the Amazon ECS console only supports registering services into private DNS namespaces. When an AWS Cloud Map private DNS namespace is created, a Route 53 private hosted zone will be created automatically.
- The VPC DNS attributes must be configured for successful DNS resolution. For information about how to configure the attributes, see DNS support in your VPC in the *Amazon VPC User Guide*.
- Amazon ECS does not support registering services into shared AWS Cloud Map namespaces.
- The DNS records created for a service discovery service always register with the private IP address for the task, rather than the public IP address, even when public namespaces are used.
- Service discovery requires that tasks specify either the `awsvpc`, `bridge`, or `host` network mode (`none` is not supported).
- If the service task definition uses the `awsvpc` network mode, you can create any combination of `A` or SRV records for each service task. If you use SRV records, a port is required.
- If the service task definition uses the `bridge` or `host` network mode, the SRV record is the only supported DNS record type. Create a SRV record for each service task. The SRV record must specify a container name and container port combination from the task definition.
- DNS records for a service discovery service can be queried within your VPC. They use the following format: `<service-discovery-service-name>.<service-discovery-namespace>`.
- When doing a DNS query on the service name, `A` records return a set of IP addresses that correspond to your tasks. SRV records return a set of IP addresses and ports for each task.
- If you have eight or fewer healthy records, Route 53 responds to all DNS queries with all of the healthy records.

- When all records are unhealthy, Route 53 responds to DNS queries with up to eight unhealthy records.
- You can configure service discovery for a service that's behind a load balancer, but service discovery traffic is always routed to the task and not the load balancer.
- Service discovery doesn't support the use of Classic Load Balancers.
- We recommend you use container-level health checks managed by Amazon ECS for your service discovery service.
  - **HealthCheckCustomConfig**—Amazon ECS manages health checks on your behalf. Amazon ECS uses information from container and health checks, and your task state, to update the health with AWS Cloud Map. This is specified using the `--health-check-custom-config` parameter when creating your service discovery service. For more information, see HealthCheckCustomConfig in the *AWS Cloud Map API Reference*.
- The AWS Cloud Map resources created when service discovery is used must be cleaned up manually.
- Tasks and instances are registered as `UNHEALTHY` until the container health checks return a value. If the health checks pass, the status is updated to `HEALTHY`. If the container health checks fail, the service discovery instance is deregistered.

Service discovery pricing

Customers using Amazon ECS service discovery are charged for Route 53 resources and AWS Cloud Map discovery API operations. This involves costs for creating the Route 53 hosted zones and queries to the service registry. For more information, see AWS Cloud Map Pricing in the *AWS Cloud Map Developer Guide*.

Amazon ECS performs container level health checks and exposes them to AWS Cloud Map custom health check API operations. This is currently made available to customers at no extra cost. If you configure additional network health checks for publicly exposed tasks, you're charged for those health checks.

Creating an Amazon ECS service that uses Service Discovery

Learn how to create a service containing a Fargate task that uses service discovery with the AWS CLI.

For a list of AWS Regions that support service discovery, see Use service discovery to connect Amazon ECS services with DNS names.

For information about the Regions that support Fargate, see the section called "AWS Fargate Regions".

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

Prerequisites

Before you start this tutorial, make sure that the following prerequisites are met:

- The latest version of the AWS CLI is installed and configured. For more information, see Installing or updating to the latest version of the AWS CLI.
- The steps described in Set up to use Amazon ECS are complete.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- You have created at least one VPC and one security group. For more information, see the section called "Create a virtual private cloud".

Step 1: Create the Service Discovery resources in AWS Cloud Map

Follow these steps to create your service discovery namespace and service discovery service:

1. Create a private Cloud Map service discovery namespace. This example creates a namespace that's called `tutorial`. Replace *vpc-abcd1234* with the ID of one of your existing VPCs.

   ```
   aws servicediscovery create-private-dns-namespace \
         --name tutorial \
         --vpc vpc-abcd1234
   ```

   The output of this command is as follows.

   ```
   {
       "OperationId": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e"
   }
   ```

2. Using the `OperationId` from the output of the previous step, verify that the private namespace was created successfully. Make note of the namespace ID because you use it in subsequent commands.

```
aws servicediscovery get-operation \
      --operation-id h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e
```

The output is as follows.

```
{
    "Operation": {
        "Id": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e",
        "Type": "CREATE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1519777852.502,
        "UpdateDate": 1519777856.086,
        "Targets": {
           "NAMESPACE": "ns-uejictsjen2i4eeg"
        }
    }
}
```

3. Using the `NAMESPACE` ID from the output of the previous step, create a service discovery service. This example creates a service named `myapplication`. Make note of the service ID and ARN because you use them in subsequent commands.

```
aws servicediscovery create-service \
      --name myapplication \
      --dns-config "NamespaceId="ns-
uejictsjen2i4eeg",DnsRecords=[{Type="A",TTL="300"}]" \
      --health-check-custom-config FailureThreshold=1
```

The output is as follows.

```
{
    "Service": {
       "Id": "srv-utcrh6wavdkggqtk",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-
utcrh6wavdkggqtk",
        "Name": "myapplication",
        "DnsConfig": {
```

```
            "NamespaceId": "ns-uejictsjen2i4eeg",
            "DnsRecords": [
                {
                    "Type": "A",
                    "TTL": 300
                }
            ]
        },
        "HealthCheckCustomConfig": {
            "FailureThreshold": 1
        },
        "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"
    }
}
```

Step 2: Create the Amazon ECS resources

Follow these steps to create your Amazon ECS cluster, task definition, and service:

1. Create an Amazon ECS cluster. This example creates a cluster that's named `tutorial`.

   ```
   aws ecs create-cluster \
         --cluster-name tutorial
   ```

2. Register a task definition that's compatible with Fargate and uses the `awsvpc` network mode. Follow these steps:

   a. Create a file that's named `fargate-task.json` with the contents of the following task definition.

      ```
      {
          "family": "tutorial-task-def",
              "networkMode": "awsvpc",
              "containerDefinitions": [
                  {
                      "name": "sample-app",
                      "image": "public.ecr.aws/docker/library/httpd:2.4",
                      "portMappings": [
                          {
                              "containerPort": 80,
                              "hostPort": 80,
                              "protocol": "tcp"
      ```

```
                    }
                ],
                "essential": true,
                "entryPoint": [
                    "sh",
                    "-c"
                ],
                "command": [
                    "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style>
 </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample
 App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a
 container in Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/
htdocs/index.html && httpd-foreground\""
                ]
            }
        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
        "memory": "512"
}
```

b. Register the task definition using `fargate-task.json`.

```
aws ecs register-task-definition \
      --cli-input-json file://fargate-task.json
```

3. Create an ECS service by following these steps:

a. Create a file that's named `ecs-service-discovery.json` with the contents of the ECS service that you're creating. This example uses the task definition that was created in the previous step. An `awsvpcConfiguration` is required because the example task definition uses the `awsvpc` network mode.

When you create the ECS service, specify the Fargate launch type, and the `LATEST` platform version that supports service discovery. When the service discovery service is created in AWS Cloud Map , `registryArn` is the ARN returned. The `securityGroups` and `subnets` must belong to the VPC that's used to create the Cloud Map namespace. You can obtain the security group and subnet IDs from the Amazon VPC Console.

```
{
    "cluster": "tutorial",
    "serviceName": "ecs-service-discovery",
    "taskDefinition": "tutorial-task-def",
    "serviceRegistries": [
       {
          "registryArn":
 "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkggqtk"
       }
    ],
    "launchType": "FARGATE",
    "platformVersion": "LATEST",
    "networkConfiguration": {
       "awsvpcConfiguration": {
          "assignPublicIp": "ENABLED",
          "securityGroups": [ "sg-abcd1234" ],
          "subnets": [ "subnet-abcd1234" ]
       }
    },
    "desiredCount": 1
}
```

b. Create your ECS service using `ecs-service-discovery.json`.

```
aws ecs create-service \
      --cli-input-json file://ecs-service-discovery.json
```

Step 3: Verify Service Discovery in AWS Cloud Map

You can verify that everything is created properly by querying your service discovery information. After service discovery is configured, you can either use AWS Cloud Map API operations, or call `dig` from an instance within your VPC. Follow these steps:

1. Using the service discovery service ID, list the service discovery instances. Make note of the instance ID (marked in bold) for resource cleanup.

```
aws servicediscovery list-instances \
      --service-id srv-utcrh6wavdkggqtk
```

The output is as follows.

```
{
    "Instances": [
        {
            "Id": "16becc26-8558-4af1-9fbd-f81be062a266",
            "Attributes": {
                "AWS_INSTANCE_IPV4": "172.31.87.2"
                "AWS_INSTANCE_PORT": "80",
                "AVAILABILITY_ZONE": "us-east-1a",
                "REGION": "us-east-1",
                "ECS_SERVICE_NAME": "ecs-service-discovery",
                "ECS_CLUSTER_NAME": "tutorial",
                "ECS_TASK_DEFINITION_FAMILY": "tutorial-task-def"
            }
        }
    ]
}
```

2. Use the service discovery namespace, service, and additional parameters such as ECS cluster name to query details about the service discovery instances.

```
aws servicediscovery discover-instances \
      --namespace-name tutorial \
      --service-name myapplication \
      --query-parameters ECS_CLUSTER_NAME=tutorial
```

3. The DNS records that are created in the Route 53 hosted zone for the service discovery service can be queried with the following AWS CLI commands:

   a. Using the namespace ID, get information about the namespace, which includes the Route 53 hosted zone ID.

   ```
   aws servicediscovery \
         get-namespace --id ns-uejictsjen2i4eeg
   ```

   The output is as follows.

   ```
   {
       "Namespace": {
           "Id": "ns-uejictsjen2i4eeg",
           "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-
   uejictsjen2i4eeg",
   ```

```
        "Name": "tutorial",
        "Type": "DNS_PRIVATE",
        "Properties": {
             "DnsProperties": {
                "HostedZoneId": "Z35JQ4ZFDRYPLV"
            }
        },
        "CreateDate": 1519777852.502,
        "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
    }
}
```

b. Using the Route 53 hosted zone ID from the previous step (see the text in bold), get the resource record set for the hosted zone.

```
aws route53 list-resource-record-sets \
      --hosted-zone-id Z35JQ4ZFDRYPLV
```

4. You can also query the DNS from an instance within your VPC using `dig`.

```
dig +short myapplication.tutorial
```

Step 4: Clean up

When you're finished with this tutorial, clean up the associated resources to avoid incurring charges for unused resources. Follow these steps:

1. Deregister the service discovery service instances using the service ID and instance ID that you noted previously.

```
aws servicediscovery deregister-instance \
      --service-id srv-utcrh6wavdkggqtk \
      --instance-id 16becc26-8558-4af1-9fbd-f81be062a266
```

The output is as follows.

```
{
    "OperationId": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv"
}
```

2. Using the `OperationId` from the output of the previous step, verify that the service discovery service instances were deregistered successfully.

```
aws servicediscovery get-operation \
      --operation-id xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv
```

```
{
  "Operation": {
        "Id": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv",
        "Type": "DEREGISTER_INSTANCE",
        "Status": "SUCCESS",
        "CreateDate": 1525984073.707,
        "UpdateDate": 1525984076.426,
        "Targets": {
            "INSTANCE": "16becc26-8558-4af1-9fbd-f81be062a266",
            "ROUTE_53_CHANGE_ID": "C5NSRG1J4I1FH",
            "SERVICE": "srv-utcrh6wavdkggqtk"
        }
    }
}
```

3. Delete the service discovery service using the service ID.

```
aws servicediscovery delete-service \
      --id srv-utcrh6wavdkggqtk
```

4. Delete the service discovery namespace using the namespace ID.

```
aws servicediscovery delete-namespace \
      --id ns-uejictsjen2i4eeg
```

   The output is as follows.

```
{
    "OperationId": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj"
}
```

5. Using the `OperationId` from the output of the previous step, verify that the service discovery namespace was deleted successfully.

```
aws servicediscovery get-operation \
```

```
      --operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj
```

The output is as follows.

```
{
    "Operation": {
        "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
        "Type": "DELETE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1525984602.211,
        "UpdateDate": 1525984602.558,
        "Targets": {
            "NAMESPACE": "ns-rymlehshst7hhukh",
            "ROUTE_53_CHANGE_ID": "CJP2A2M86XW3O"
        }
    }
}
```

6. Update the desired count for the Amazon ECS service to `0`. You must do this to delete the service in the next step.

```
aws ecs update-service \
      --cluster tutorial \
      --service ecs-service-discovery \
      --desired-count 0
```

7. Delete the Amazon ECS service.

```
aws ecs delete-service \
      --cluster tutorial \
      --service ecs-service-discovery
```

8. Delete the Amazon ECS cluster.

```
aws ecs delete-cluster \
      --cluster tutorial
```

Use Amazon VPC Lattice to connect, observe, and secure your Amazon ECS services

Amazon VPC Lattice is a fully managed application networking service that enables Amazon ECS customers to observe, secure, and monitor applications built across AWS compute services, VPCs, and accounts—without requiring any code changes.

VPC Lattice uses target groups, which are a collection of compute resources. These targets run your application or service and can be Amazon EC2 instances, IP addresses, Lambda functions, and Application Load Balancers. By associating their Amazon ECS services with a VPC Lattice target group, customers can now enable Amazon ECS tasks as IP targets in VPC Lattice. Amazon ECS automatically registers tasks to the VPC Lattice target group when tasks for the registered service are launched.

> **Note**
>
> When using five VPC Lattice configurations, your deployment time may be slightly longer than when using fewer configurations.

A listener rule is used to forward traffic to a specified target group when the conditions are met. A listener checks for connection requests using the protocol on the port you configured. A service routes requests to it's registered targets based on the rules that you define when you configured your listener.

Amazon ECS also automatically replaces a task if it becomes unhealthy according to VPC Lattice health checks. Once associated with VPC Lattice, Amazon ECS customers can also take advantage of many other cross-compute connectivity, security, and observability features in VPC Lattice like connecting to services across clusters, VPCs, and accounts with AWS Resource Access Manager, IAM integration for authorization and authentication, and advanced traffic management features.

Amazon ECS customers can benefit from VPC Lattice in the following ways.

- Increased developer productivity - VPC Lattice boosts developer productivity by letting you focus on building features, while VPC Lattice handles networking, security and observability challenges in a uniform way across all compute platforms.
- Better security posture - VPC Lattice enables your developers to easily authenticate and secure communication across applications and compute platforms, enforce encryption in transit, and

apply granular access controls with VPC Lattice Auth policies. This allows you to adopt a stronger security posture that meets industry leading regulatory and compliance requirements.

- Improved application scalability and resilience - VPC Lattice lets you create a network of deployed applications with features like path, header, and method-based routing, authentication, authorization, and monitoring. These benefits are provided with no resource overhead on workloads and can support multi-cluster deployments that generate millions of requests per second without adding significant latency.
- Deployment flexibility with heterogeneous infrastructure - VPC Lattice provides consistent features across all compute services like Amazon ECS, Fargate, Amazon EC2, Amazon EKS, and Lambda and allows your organization the flexibility to choose suitable infrastructure for each application.

How VPC Lattice works with other Amazon ECS services

Using VPC Lattice with Amazon ECS may change the way you use other Amazon ECS services, while others stay the same.

Application Load Balancers

You no longer need to create a specific Application Load Balancer to use with the Application Load Balancer target group type in VPC Lattice that then links to the Amazon ECS service. You only need to configure your Amazon ECS service with a VPC Lattice target group instead. You can also still choose to use Application Load Balancer with Amazon ECS at the same time.

Amazon ECS rolling deployments

Only Amazon ECS rolling deployments work with VPC Lattice, and Amazon ECS safely brings tasks into and removes them from services during deployment. Code deploy and Blue/Green deployments aren't supported.

To learn more about VPC Lattice, see the [Amazon VPC Lattice User Guide](https://docs.aws.amazon.com/vpc-lattice/latest/ug/).

Create a service that uses VPC Lattice

You can use either the AWS Management Console or the AWS CLI to create a service with VPC Lattice.

Prerequisites

Before you start this tutorial, make sure that the following prerequisites are met:

- The latest version of the AWS CLI is installed and configured. For more information, see Installing the AWS Command Line Interface.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

- The steps described in Set up to use Amazon ECS are complete.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.

Create a service that uses VPC Lattice with the AWS Management Console

Follow these steps to create a service with VPC Lattice using the AWS Management Console.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation page, choose **Clusters**.
3. On the **Clusters** page, choose the cluster to create the service in.
4. From the **Services** tab, choose **Create**.

   If you've never created a service before, follow the steps found in Creating an Amazon ECS service using the console, then continue with these steps when you reach the VPC Lattice section.
5. Choose to **Turn on VPC Lattice** by checking the button.
6. To use an existing role, for **ECS infrastructure role for Amazon ECS**, choose one that you've already created to use when creating the VPC Lattice target group. To create a new role, **Create ECS infrastructure role**.
7. Choose the **VPC**.

   The **VPC** depends on the networking mode you selected when you registered your task definition. If you use the `host` or `network` mode with the EC2 launch type, choose your VPC.

   For the `awsvpc` mode, the VPC is automatically selected based on the VPC you chose under **Networking** and can't be changed.

8. Under **Target Groups** choose the target group or groups. You need to choose at least one target group and can have a maximum of five. Choose **Add target group** to add additional target groups. Choose the **Port name**, **Protocol**, and **Port** for each target group you chose. To delete a target group, choose **Remove**.

> **Note**
>
> - If you want to add existing target groups, you need use the AWS CLI. For instructions on how to add target groups using the AWS CLI, see register-targets in the *AWS Command Line Interface Reference*.
> - While a VPC Lattice service can have multiple target groups, each target group can only be added to one service.

9. At this point, you navigate to the VPC Lattice console to continue setting up. This is where you include your new target groups in the listener default action or in the rules of an existing VPC Lattice service.

   For more information, see Listener rules for your VPC Lattice service.

> **Important**
>
> You need to allow the inbound rule `vpc-lattice` prefix to your security group or tasks and health checks can fail.

Create a service that uses VPC Lattice with the AWS CLI

Use the AWS CLI to create a service with VPC Lattice. Replace each *user input placeholder* with your own information.

1. Create a target group configuration file. The following example is named `tg-config.json`

```
{
    "ipAddressType": "IPV4",
    "port": 443,
    "protocol": "HTTPS",
    "protocolVersion": "HTTP1",
    "vpcIdentifier": "vpc-f1663d9868EXAMPLE"
```

```
}
```

2. Use the following command to create a VPC Lattice target group.

```
aws vpc-lattice create-target-group \
    --name my-lattice-target-group-ip \
    --type IP \
    --config file://tg-config.json
```

Example output:

```
{
    "arn": "arn:aws:vpc-lattice:us-east-2:123456789012:targetgroup/
tg-0eaa4b9ab4EXAMPLE",
    "config": {
        "healthCheck": {
            "enabled": true,
            "healthCheckIntervalSeconds": 30,
            "healthCheckTimeoutSeconds": 5,
            "healthyThresholdCount": 5,
            "matcher": {
                "httpCode": "200"
            },
            "path": "/",
            "protocol": "HTTPS",
            "protocolVersion": "HTTP1",
            "unhealthyThresholdCount": 2
        },
        "ipAddressType": "IPV4",
        "port": 443,
        "protocol": "HTTPS",
        "protocolVersion": "HTTP1",
        "vpcIdentifier": "vpc-f1663d9868EXAMPLE"
    },
    "id": "tg-0eaa4b9ab4EXAMPLE",
    "name": "my-lattice-target-group-ip",
    "status": "CREATE_IN_PROGRESS",
    "type": "IP"
}
```

3. The following JSON file named *`ecs-service-vpc-lattice.json`* is an example used to attach an Amazon ECS service to a VPC Lattice target group. The `portName` in the example

below is the same one you defined in your task definition's `portMappings` property's `name` field.

```
{
    "serviceName": "ecs-service-vpc-lattice",
    "taskDefinition": "ecs-task-def",
        "vpcLatticeConfigurations": [
        {
            "targetGroupArn": "arn:aws:vpc-lattice:us-
west-2:123456789012:targetgroup/tg-0eaa4b9ab4EXAMPLE",
            "portName": "testvpclattice",
            "roleArn": "arn:aws:iam::123456789012:role/
ecsInfrastructureRoleVpcLattice"
        }
    ],
    "desiredCount": 5,
    "role": "ecsServiceRole"
}
```

Use the following command to create an Amazon ECS service and attach it to the VPC Lattice target group using the json example above.

```
aws ecs create-service \
    --cluster clusterName \
    --serviceName ecs-service-vpc-lattice \
    --cli-input-json file://ecs-service-vpc-lattice.json
```

> **Note**
>
> VPC Lattice isn't supported on Amazon ECS Anywhere.

Protect your Amazon ECS tasks from being terminated by scale-in events

You can use Amazon ECS task scale-in protection to protect your tasks from being terminated by scale-in events from either service auto scaling or deployments.

Certain applications require a mechanism to safeguard mission-critical tasks from termination by scale-in events during times of low utilization or during service deployments. For example:

- You have a queue-processing asynchronous application such as a video transcoding job where some tasks need to run for hours even when cumulative service utilization is low.
- You have a gaming application that runs game servers as Amazon ECS tasks that need to continue running even if all users have logged-out to reduce start-up latency of a server reboot.
- When you deploy a new code version, you need tasks to continue running because it would be expensive to reprocess.

To protect tasks that belong to your service from terminating in a scale-in event, set the `ProtectionEnabled` attribute to `true`. When you set `ProtectionEnabled` to true, tasks are protected for 2 hours by default. You can then customize the protection period by using the `ExpiresInMinutes` attribute. You can protect your tasks for a minimum of 1 minute and up to a maximum of 2880 minutes (48 hours). If you're using the AWS CLI, you can specify the `--protection-enabled` option.

After a task finishes its requisite work, you can set the `ProtectionEnabled` attribute to `false`, allowing the task to be terminated by subsequent scale-in events. If you're using the AWS CLI, you can specify the `--no-protection-enabled` option.

Task scale-in protection mechanisms

You can set and get task scale-in protection using either the Amazon ECS container agent endpoint or the Amazon ECS API.

- **Amazon ECS container agent endpoint**

  We recommend using the Amazon ECS container agent endpoint for tasks that can self-determine the need to be protected. Use this approach for queue-based or job-processing workloads.

  When a container starts processing work, for example by consuming an SQS message, you can set the `ProtectionEnabled` attribute through the task scale-in protection endpoint path `$ECS_AGENT_URI/task-protection/v1/state` from within the container. Amazon ECS will not terminate this task during scale-in events. After your task finishes its work, you can clear the `ProtectionEnabled` attribute using the same endpoint, making the task eligible for termination during subsequent scale-in events.

For more information about the Amazon ECS container agent endpoint, see Amazon ECS task scale-in protection endpoint.

- **Amazon ECS API**

  You can use the Amazon ECS API to set and retrieve task scale-in protection if your application has a component that tracks the status of active tasks. Use `UpdateTaskProtection` to mark one or more tasks as protected. Use `GetTaskProtection` to retrieve the protection status.

  An example of this approach would be if your application is hosting game server sessions as Amazon ECS tasks. When a user logs in to a session on the server (task), you can mark the task as protected. After the user logs out, you can either clear the protection specifically for this task or periodically clear protection for similar tasks that no longer have active sessions, depending on your requirement to keep idle servers.

  For more information, see UpdateTaskProtection and GetTaskProtection in the *Amazon Elastic Container Service API Reference*.

You can combine both approaches. For example, use the Amazon ECS agent endpoint to set task protection from within a container and use the Amazon ECS API to remove task protection from your external controller service.

Considerations

Consider the following points before using task scale-in protection:

- Task scale-in protection is only supported with tasks deployed from a service.
- We recommend using the Amazon ECS container agent endpoint because the Amazon ECS agent has built-in retry mechanisms and a simpler interface.
- You can reset the task scale-in protection expiration period by calling `UpdateTaskProtection` for a task that already has protection turned on.
- Determine how long a task would need to complete its requisite work and set the `expiresInMinutes` property accordingly. If you set the protection expiration longer than necessary, then you will incur costs and face delays in the deployment of new tasks.
- Task scale-in protection is supported on Amazon ECS container agent `1.65.0` or later.

You can add support for this feature on Amazon EC2 instances using older versions of the Amazon ECS container agent by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent](#).

- Deployment considerations:
  - If the service uses a rolling update, new tasks will be created but tasks running older version will not be terminated until `protectionEnabled` is cleared or expires. You can adjust the `maximumPercentage` parameter in deployment configuration to a value that allows new tasks to be created when old tasks are protected.
  - If a blue/green update is applied, the blue deployment with protected tasks will not be removed if tasks have `protectionEnabled`. Traffic will be diverted to the new tasks that come up and older tasks will only be removed when `protectionEnabled` is cleared or expires. Depending on the timeout of the CodeDeploy or CloudFormation updates, the deployment may timeout and the older Blue tasks may still be present.
  - If you use CloudFormation, the update-stack has a 3 hour timeout. Therefore, if you set your task protection for longer than 3 hours, then your CloudFormation deployment may result in failure and rollback.

    During the time your old tasks are protected, the CloudFormation stack shows `UPDATE_IN_PROGRESS`. If task scale-in protection is removed or expires within the 3 hour window, your deployment will succeed and move to the `UPDATE_COMPLETE` status. If the deployment is stuck in `UPDATE_IN_PROGRESS` for more than 3 hours, it will fail and show `UPDATE_FAILED` state, and will then be rolled back to old task set.
  - Amazon ECS sends service events when protected tasks keep a deployment (rolling or blue/green) from reaching the steady state, so that you can take remedial actions. While trying to update the protection status of a task, if you receive a `DEPLOYMENT_BLOCKED` error message, it means the service has more protected tasks than the desired count of tasks for the service. To resolve this error, do one the following:
    - Wait for the current task protection to expire. Then set task protection.
    - Determine which tasks can be stopped. Then use `UpdateTaskProtection`with the `protectionEnabled` option set to `false` for these tasks.
    - Increase the desired task count of the service to more than the number of protected tasks.

IAM permissions required for task scale-in protection

The task must have the Amazon ECS task role with the following permissions:

- `ecs:GetTaskProtection`: Allows the Amazon ECS container agent to call `GetTaskProtection`.
- `ecs:UpdateTaskProtection`: Allows the Amazon ECS container agent to call `UpdateTaskProtection`.

Amazon ECS task scale-in protection endpoint

The Amazon ECS container agent automatically injects the `ECS_AGENT_URI` environment variable into the containers of Amazon ECS tasks to provide a method to interact with the container agent API endpoint.

We recommend using the Amazon ECS container agent endpoint for tasks that can self-determine the need to be protected.

When a container starts processing work, you can set the `protectionEnabled` attribute using the task scale-in protection endpoint path `$ECS_AGENT_URI/task-protection/v1/state` from within the container.

Use a PUT request to this URI from within a container to set task scale-in protection. A GET request to this URI returns the current protection status of a task.

Task scale-in protection request parameters

You can set task scale-in protection using the `${ECS_AGENT_URI}/task-protection/v1/state` endpoint with the following request parameters.

`ProtectionEnabled`

Specify `true` to mark a task for protection. Specify `false` to remove protection and make the task eligible for termination.

Type: Boolean

Required: Yes

`ExpiresInMinutes`

The number of minutes the task is protected. You can specify a minimum of 1 minute to up to 2,880 minutes (48 hours). During this time period, your task will not be terminated by scale-in events from service Auto Scaling or deployments. After this time period lapses, the `protectionEnabled` parameter is set to `false`.

If you don’t specify the time, then the task is automatically protected for 120 minutes (2 hours).

Type: Integer

Required: No

The following examples show how to set task protection with different durations.

Example of how to protect a task with the default time period

This example shows how to protect a task with the default time period of 2 hours.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-protection/v1/state --data '{"ProtectionEnabled":true}'
```

Example of how to protect a task for 60 minutes

This example shows how to protect a task for 60 minutes using the `expiresInMinutes` parameter.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":60}'
```

Example of how to protect a task for 24 hours

This example shows how to protect a task for 24 hours using the `expiresInMinutes` parameter.

```
curl --request PUT --header 'Content-Type: application/json' ${ECS_AGENT_URI}/task-protection/v1/state --data '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}'
```

Examples for Windows containers

For Windows containers, you can use PowerShell's `Invoke-RestMethod` cmdlet instead of curl. The following examples show the PowerShell equivalents of the previous curl commands.

Example of how to protect a Windows container task with the default time period

This example shows how to protect a task with the default time period of 2 hours using PowerShell.

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true}' -ContentType 'application/json'
```

Example of how to protect a Windows container task for 60 minutes

This example shows how to protect a task for 60 minutes using the `expiresInMinutes` parameter with PowerShell.

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true,"ExpiresInMinutes":60}' -ContentType 'application/json'
```

Example of how to protect a Windows container task for 24 hours

This example shows how to protect a task for 24 hours using the `expiresInMinutes` parameter with PowerShell.

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Put -Body
 '{"ProtectionEnabled":true,"ExpiresInMinutes":1440}' -ContentType 'application/json'
```

The PUT request returns the following response.

```
{
  "protection": {
    "ExpirationDate": "2023-12-20T21:57:44.837Z",
    "ProtectionEnabled": true,
    "TaskArn": "arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
  }
}
```

Task scale-in protection response parameters

The following information is returned from the task scale-in protection endpoint `${ECS_AGENT_URI}/task-protection/v1/state` in the JSON response.

`ExpirationDate`

The epoch time when protection for the task will expire. If the task is not protected, this value is null.

`ProtectionEnabled`

The protection status of the task. If scale-in protection is enabled for a task, the value is `true`. Otherwise, it is `false`.

`TaskArn`

The full Amazon Resource Name (ARN) of the task that the container belongs to.

The following example shows the details returned for a protected task.

```
curl --request GET ${ECS_AGENT_URI}/task-protection/v1/state
```

For Windows containers, use the following PowerShell command to get the protection status:

```
Invoke-RestMethod -Uri $env:ECS_AGENT_URI/task-protection/v1/state -Method Get
```

```
{
    "protection":{
        "ExpirationDate":"2023-12-20T21:57:44Z",
        "ProtectionEnabled":true,
        "TaskArn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0"
    }
}
```

The following information is returned when a failure occurs.

`Arn`

The full Amazon Resource Name (ARN) of the task.

`Detail`

The details related to the failure.

`Reason`

The reason for the failure.

The following example shows the details returned for a task that is not protected.

```
{
    "failure":{
        "Arn":"arn:aws:ecs:us-west-2:111122223333:task/1234567890abcdef0",
        "Detail":null,
        "Reason":"TASK_NOT_VALID"
    }
}
```

The following information is returned when an exception occurs.

`requestID`

The AWS request ID for the Amazon ECS API call that results in an exception.

`Arn`

The full Amazon Resource Name (ARN) of the task or service.

`Code`

The error code.

`Message`

The error message.

> **Note**
>
> If a `RequestError` or `RequestTimeout` error appears, it is likely that it's a networking issue. Try using VPC endpoints for Amazon ECS.

The following example shows the details returned when an error occurs.

```
{
    "requestID":"12345-abc-6789-0123-abc",
    "error":{
        "Arn":"arn:aws:ecs:us-west-2:555555555555:task/my-cluster-
name/1234567890abcdef0",
        "Code":"AccessDeniedException",
        "Message":"User: arn:aws:sts::444455556666:assumed-role/my-ecs-task-
role/1234567890abcdef0 is not authorized to perform: ecs:GetTaskProtection on resource:
```

```
 arn:aws:ecs:us-west-2:555555555555:task/test/1234567890abcdef0 because no identity-
based policy allows the ecs:GetTaskProtection action"
    }
}
```

The following error appears if the Amazon ECS agent is unable to get a response from the Amazon ECS endpoint for reasons such as network issues or the Amazon ECS control plane is down.

```
{
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "RequestCanceled",
    "Message": "Timed out calling Amazon ECS Task Protection API"
  }
}
```

The following error appears when the Amazon ECS agent gets a throttling exception from Amazon ECS.

```
{
  "requestID": "12345-abc-6789-0123-abc",
  "error": {
    "Arn": "arn:aws:ecs:us-west-2:555555555555:task/my-cluster-name/1234567890abcdef0",
    "Code": "ThrottlingException",
    "Message": "Rate exceeded"
  }
}
```

Use fault injection with your Amazon ECS and Fargate workloads

Customers can utilize fault injection with Amazon ECS on both Amazon EC2 and Fargate to test how their application responds to certain impairment scenarios. These tests provide information you can use to optimize your application's performance and resiliency.

When fault injection is enabled, the Amazon ECS container agent allows tasks access to new fault injection endpoints. You need to opt-in in order to use fault injection by setting the `enableFaultInjection` task definition parameter value to `true`. The default value is `false`.

```
{
    ...
```

```
    "enableFaultInjection": true
}
```

> **Note**
>
> Fault injection only works with tasks using the `awsvpc` or `host` network modes. Fault injection isn't available on Windows.

For information on how to enable fault injection in the AWS Management Console, see Creating an Amazon ECS task definition using the console.

You'll need to enable the feature for testing in the AWS Fault Injection Service. For more information, see Use the AWS FIS aws:ecs:task actions.

> **Note**
>
> If you don't use thenew Amazon ECS optimized AMIs, or have a custom AMI, install the following dependencies:
>
> - `tc`
> - `sch_netem` kernel module

Amazon ECS fault injection endpoints

The Amazon ECS container agent automatically injects the `ECS_AGENT_URI` environment variable into the containers of Amazon ECS tasks to provide a method to interact with the container agent API endpoint. Each endpoint includes a `/start`, `/stop`, and `/status` endpoint. The endpoints only accept requests from tasks that have enabled fault injection, and each endpoint has a rate limit of **1** request per **5** seconds per container. Exceeding this limit results in an error.

> **Note**
>
> Amazon ECS Agent `version 1.88.0+` is required to use the fault injection endpoints.

The three endpoints for use with fault injection are:

- Network blackhole port endpoint
- Network packet loss endpoint
- Network latency endpoint

A successful request results in a response code of `200` with a message of `running` when you call the `/start` endpoint, `stopped` for the `/stop` endpoint, and `running` or `not-running` for the `/status` endpoint.

```
{
    "Status": <string>
}
```

An unsuccessful request returns one of the follow error codes:

- `400` - Bad request
- `409` - Fault injection request conflicts with another running fault
- `429` - Request was throttled
- `500` - Server had an unexpected error

```
{
 "Error":  <string message>
}
```

Note

Either one network latency fault or one network packet loss fault can be injected at a time. Trying to inject more than one results in the request being rejected.

Network blackhole port endpoint

The `{ECS_AGENT_URI}/fault/v1/network-blackhole-port` endpoint drops inbound or outbound traffic for a specific port and protocol in a task's network namespace and is compatible with two modes:

- **awsvpc** - the changes are applied to the task network namespace

- **host** - the changes are applied to the default network namespace container instance

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/start

This endpoint starts the network blackhole port fault injections and has the following parameters:

Port

The specified port to use for the blackhole port fault injection.

Type: Integer

Required: Yes

Protocol

The protocol to use for the blackhole port fault injection.

Type: String

Valid values: `tcp | udp`

Required: Yes

TrafficType

The traffic type used by the fault injection.

Type: String

Valid values: `ingress | egress`

Required: Yes

SourcesToFilter

A JSON array of IPv4 addresses or CIDR blocks that are protected from the fault.

Type: Array of strings

Required: No

The following is an example request for using the `start` endpoint (replace the *`red`* values with your own):

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/start

Http method:POST

Request payload:
{
    "Port": 1234,
    "Protocol": "tcp|udp",
    "TrafficType": "ingress|egress"
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/stop

This endpoint stops the fault specified in the request. This endpoint has the following parameters:

Port

The port impacted by the fault that should be stopped.

Type: Integer

Required: Yes

Protocol

The protocol to use to stop the fault.

Type: String

Valid values: `tcp | udp`

Required: Yes

TrafficType

The traffic type used by the fault injection.

Type: String

Valid values: `ingress | egress`

Required: Yes

The following is an example request for using the `stop` endpoint (replace the *red* values with your own):

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/stop

Http method: POST

Request payload:
{
    "Port": 1234,
    "Protocol": "tcp|udp",
    "TrafficType": "ingress|egress",
}
```

{ECS_AGENT_URI}/fault/v1/network-blackhole-port/status

This endpoint is used to check the status of the fault injection. This endpoint has the following parameters:

Port

The impacted port to check for the fault's status.

Type: Integer

Required: Yes

Protocol

The protocol to use when checking for the fault's status.

Type: String

Valid values: `tcp` | `udp`

Required: Yes

TrafficType

The traffic type used by the fault injection.

Type: String

Valid values: `ingress | egress`

Required: Yes

The following is an example request for using the `status` endpoint (replace the *red* values with your own):

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-blackhole-port/status

Http method: POST

Request payload:
{
   "Port": 1234,
   "Protocol": "tcp|udp",
   "TrafficType": "ingress|egress",
}
```

Network latency endpoint

The `{ECS_AGENT_URI}/fault/v1/network-latency` endpoint adds delay and jitter to the task's network interface for traffic to a specific sources. The endpoint is compatible with two modes:

- **awsvpc** - the changes are applied to the task network interface
- **host** - the changes are applied to the default network interface

{ECS_AGENT_URI}/fault/v1/network-latency/start

This `/start` endpoint begins the network latency fault injection and has the following parameters:

DelayMilliseconds

The number of milliseconds of delay to add to the network interface to use for the fault injection.

Type: Integer

Required: Yes

JitterMilliseconds

The number of milliseconds of jitter to add to the network interface to use for the fault injection.

Type: Integer

Required: Yes

Sources

A JSON array of IPv4 addresses or CIDR blocks that are destination for use with fault injection.

Type: Array of strings

Required: Yes

SourcesToFilter

A JSON array of IPv4 addresses or CIDR blocks that are protected from the fault. `SourcesToFilter` takes priority over `Sources`.

Type: Array of strings

Required: No

The following is an example request for using the `/start` endpoint (replace the *red* values with your own):

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/start

Http method: POST

Request payload:
{
    "DelayMilliseconds": 123,
    "JitterMilliseconds": 123,
    "Sources": ["${IP1}", "${IP2}", ...],
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-latency/stop and /status

The `{ECS_AGENT_URI}/fault/v1/network-latency/stop` endpoint stops the fault, and the `{ECS_AGENT_URI}/fault/v1/network-latency/status` checks the fault's status.

The following are two example requests for using the `/stop` and the `/status` endpoints. Both use the `POST HTTP` method.

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/stop
```

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-latency/status
```

Network packet loss endpoint

The `{ECS_AGENT_URI}/fault/v1/network-packet-loss` endpoint adds packet loss to the given network interface. This endpoint is compatible with two modes:

- **awsvpc** - the changes are applied to the task network interface
- **host** - the changes are applied to the default network interface

{ECS_AGENT_URI}/fault/v1/network-packet-loss/start

This `/start` endpoint begins the network packet loss fault injection and has the following parameters:

LossPercent

The percentage of packet loss

Type: Integer

Required: Yes

Sources

A JSON array of IPv4 addresses or CIDR blocks to use for the fault injection tests.

Type: Array of strings

Required: Yes

SourcesToFilter

A JSON array of IPv4 addresses or CIDR blocks that are protected from the fault. `SourcesToFilter` takes priority over `Sources`.

Type: Array of strings

Required: No

The following is an example request for using the `start` endpoint (replace the *red* values with your own):

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-packet-loss/start

Http method: POST

{
    "LossPercent": 6,
    "Sources": ["${IP1}", "${IP2}", ...],
    "SourcesToFilter": ["${IP1}", "${IP2}", ...],
}
```

{ECS_AGENT_URI}/fault/v1/network-packet-loss/stop and /status

The `{ECS_AGENT_URI}/fault/v1/network-packet-loss/stop` endpoint stops the fault, and the `{ECS_AGENT_URI}/fault/v1/network-packet-loss/status` checks the fault's status. Only one of each type of fault is supported at a time.

The following are two example requests for using the `/stop` and the `/status` endpoints. Both use the `POST HTTP` method.

```
Endpoint: ${ECS_AGENT_URI}/fault/v1/network-packet-loss/stop
```

```
Endpoint: ${{ECS_AGENT_URI}/fault/v1/network-packet-loss/status
```

Update Amazon ECS service parameters

After you create a service, there are times when you might need to update the service parameters, for example, the number of tasks.

When the service scheduler launches new tasks, it determines task placement in your cluster with the following logic.

- Determine which of the container instances in your cluster can support your service's task definition. For example, they have the required CPU, memory, ports, and container instance attributes.
- By default, the service scheduler attempts to balance tasks across Availability Zones in this manner even though you can choose a different placement strategy.

- Sort the valid container instances by the fewest number of running tasks for this service in the same Availability Zone as the instance. For example, if zone A has one running service task and zones B and C each have zero, valid container instances in either zone B or C are considered optimal for placement.
- Place the new service task on a valid container instance in an optimal Availability Zone (based on the previous steps), favoring container instances with the fewest number of running tasks for this service.

When the service scheduler stops running tasks, it attempts to maintain balance across the Availability Zones in your cluster using the following logic:

- Sort the container instances by the largest number of running tasks for this service in the same Availability Zone as the instance. For example, if zone A has one running service task and zones B and C each have two, container instances in either zone B or C are considered optimal for termination.
- Stop the task on a container instance in an optimal Availability Zone (based on the previous steps), favoring container instances with the largest number of running tasks for this service.

Use the list to determine if you can change the service parameter.

Availability Zone rebalancing

Indicates whether to use Availability Zone rebalancing for the service.

You can change this parameter for rolling deployments.

Capacity provider strategy

The details of a capacity provider strategy. You can set a capacity provider when you create a cluster, run a task, or update a service.

When you use Fargate, the capacity providers are `FARGATE` or `FARGATE_SPOT`.

When you use Amazon EC2, the capacity providers are Auto Scaling groups.

You can change capacity providers for rolling deployments and blue/green deployments.

The following list provides the valid transitions:

- Update the Fargate launch type to an Auto Scaling group capacity provider.

- Update the Amazon EC2 launch type to a Fargate capacity provider.
- Update the Fargate capacity provider to an Auto Scaling group capacity provider.
- Update the Amazon EC2 capacity provider to a Fargate capacity provider.
- Update the Auto Scaling group or Fargate capacity provider back to the launch type. When you use the CLI, or API, you pass an empty list in the `capacityProviderStrategy` parameter.

Cluster

You can't change the cluster name.

Deployment configuration

The deployment configuration includes the CloudWatch alarms, and circuit breaker used to detect failures and the required configuration.

The deployment circuit breaker determines whether a service deployment will fail if the service can't reach a steady state. If you use the deployment circuit breaker, a service deployment will transition to a failed state and stop launching new tasks. If you use the rollback option, when a service deployment fails, the service is rolled back to the last deployment that completed successfully.

When you update a service that uses Amazon ECS circuit breaker, Amazon ECS creates a service deployment and a service revision. These resources allow you to view detailed information about the service history. For more information, see View service history using Amazon ECS service deployments.

The service scheduler uses the minimum healthy percent and maximum percent parameters (in the deployment configuration for the service) to determine the deployment strategy.

If a service uses the rolling update (ECS) deployment type, the **minimum healthy percent** represents a lower limit on the number of tasks in a service that must remain in the `RUNNING` state during a deployment, as a percentage of the desired number of tasks (rounded up to the nearest integer). The parameter also applies while any container instances are in the `DRAINING` state if the service contains tasks using the EC2 launch type. Use this parameter to deploy without using additional cluster capacity. For example, if your service has a desired number of four tasks and a minimum healthy percent of 50 percent, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. The service considers tasks healthy for services that do not use a load balancer if they are in the `RUNNING` state. The service considers tasks healthy for services that do use a load balancer if they are in the `RUNNING` state and they are reported as healthy by the load balancer. The default value for minimum healthy percent is 100 percent.

If a service uses the rolling update (ECS) deployment type, the **maximum percent** parameter represents an upper limit on the number of tasks in a service that are allowed in the PENDING, RUNNING, or STOPPING state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer). The parameter also applies while any container instances are in the DRAINING state if the service contains tasks using the EC2 launch type. Use this parameter to define the deployment batch size. For example, if your service has a desired number of four tasks and a maximum percent value of 200 percent, the scheduler may start four new tasks before stopping the four older tasks. This is provided that the cluster resources required to do this are available. The default value for the maximum percent is 200 percent.

When the service scheduler replaces a task during an update, the service first removes the task from the load balancer (if used) and waits for the connections to drain. Then, the equivalent of **docker stop** is issued to the containers running in the task. This results in a SIGTERM signal and a 30-second timeout, after which SIGKILL is sent and the containers are forcibly stopped. If the container handles the SIGTERM signal gracefully and exits within 30 seconds from receiving it, no SIGKILL signal is sent. The service scheduler starts and stops tasks as defined by your minimum healthy percent and maximum percent settings.

The service scheduler also replaces tasks determined to be unhealthy after a container health check or a load balancer target group health check fails. This replacement depends on the maximumPercent and desiredCount service definition parameters. If a task is marked unhealthy, the service scheduler will first start a replacement task. Then, the following happens.

- If the replacement task has a health status of HEALTHY, the service scheduler stops the unhealthy task
- If the replacement task has a health status of UNHEALTHY, the scheduler will stop either the unhealthy replacement task or the existing unhealthy task to get the total task count to equal desiredCount.

If the maximumPercent parameter limits the scheduler from starting a replacement task first, the scheduler will stop an unhealthy task one at a time at random to free up capacity, and then start a replacement task. The start and stop process continues until all unhealthy tasks are replaced with healthy tasks. Once all unhealthy tasks have been replaced and only healthy tasks are running, if the total task count exceeds the desiredCount, healthy tasks are stopped at random until the total task count equals desiredCount. For more information about maximumPercent and desiredCount, see [Service definition parameters](#).

Deployment controller

The deployment controller to use for the service. There are three deployment controller types available:

- `ECS`
- `EXTERNAL`
- `CODE_DEPLOY`

When you update a service, you can update the deployment controller it uses. The following list provides the valid transitions:

- Update from CodeDeploy blue/green deployments (`CODE_DEPLOY`) to ECS rolling or blue/green deployments (`ECS`).
- Update from CodeDeploy blue/green deployments (`CODE_DEPLOY`) to external deployments (`EXTERNAL`).
- Update from ECS rolling or blue/green deployments (`ECS`) to external deployments (`EXTERNAL`).
- Update from external deployments (`EXTERNAL`) to ECS rolling or blue/green deployments (`ECS`).

Consider the following when you update a service's deployment controller:

- You can't update the deployment controller of a service from the `ECS` deployment controller to any of the other controllers if it uses VPC Lattice or Amazon ECS Service Connect.
- You can't update the deployment controller of a service during an ongoing service deployment.
- You can't update the deployment controller of a service to `CODE_DEPLOY` if there are no load balancers on the service.
- You can't update the deployment controller of a service from `ECS` to any of the other controllers if the `deploymentConfiguration` includes alarms, a deployment circuit breaker, or a `BLUE_GREEN` deployment strategy. For more information, see [Amazon ECS service deployment controllers and strategies](#).
- The value you specify for `versionConsistency` in the container definition won't be used by Amazon ECS if you update the deployment controller of the service from `ECS` to any of the other controllers.
- If you update a service's deployment controller from `ECS` to any of the other controllers, the `UpdateService` and `DescribeService` API responses will still return `deployments`

instead of `taskSets`. For more information about `UpdateService` and `CreateService`, see [UpdateService](#) and [CreateService](#) in the *Amazon ECS API Reference*.

- If a service uses a rolling update deployment strategy, updating the deployment controller from `ECS` to any of the other controllers will change how the `maximumPercent` value in the `deploymentConfiguration` is used. Instead of just being used as a cap on total tasks in a rolling update deployment, `maximumPercent` is used for replacing unhealthy tasks. For more information on how the scheduler replaces unhealthy tasks, see [Amazon ECS services](#).
- If you update a service's deployment controller from `ECS` to any of the other deployment controllers, any `advancedConfiguration` that you specify with your load balancer configuration will be ignored. For more information, see [LoadBalancer](#) and [AdvancedConfiguration](#) in the *Amazon ECS API reference*.

When updating the deployment controller for a service using CloudFormation, consider the following depending on the type of migration you're performing.

- If you have a CloudFormation template that contains the `EXTERNAL` deployment controller information as well as `TaskSet` and `PrimaryTaskSet` resources, and you remove the task set resources from the template when updating from `EXTERNAL` to `ECS`, the `DescribeTaskSet` and `DeleteTaskSet` API calls will return a 400 error after the deployment controller is updated to `ECS`. This results in a CloudFormation delete failure on the task set resources, even though the CloudFormation stack transitions to `UPDATE_COMPLETE` status. For more information, see [Resource removed from stack but not deleted](#) in the AWS CloudFormation User Guide. To fix this issue, delete the task sets directly using the Amazon ECS `DeleteTaskSet` API. For more information about how to delete a task set, see [DeleteTaskSet](#) in the *Amazon Elastic Container Service API Reference*.
- If you're migrating from `CODE_DEPLOY` to `ECS` with a new task definition and CloudFormation performs a rollback operation, the Amazon ECS `UpdateService` request fails with the following error:

```
Resource handler returned message: "Invalid request provided: Unable to update
task definition on services with a CODE_DEPLOY deployment controller. Use AWS
CodeDeploy to trigger a new deployment. (Service: Ecs, Status Code: 400,
Request ID: 0abda1e2-f7b3-4e96-b6e9-c8bc585181ac) (SDK Attempt Count: 1)"
(RequestToken: ba8767eb-c99e-efed-6ec8-25011d9473f0, HandlerErrorCode:
 InvalidRequest)
```

- After a successful migration from `ECS` to `EXTERNAL` deployment controller, you need to manually remove the `ACTIVE` task set, because Amazon ECS no longer manages the deployment. For information about how to delete a task set, see [DeleteTaskSet] in the Amazon Elastic Container Service *API Reference*.

Desired task count

The number of instantiations of the task to place and keep running in your service.

If you want to temporarily stop your service, set this value to 0. Then, when you are ready to start the service, update the service with the original value.

You can change this parameter for rolling deployments, and blue/green deployments.

Enable managed tags

Determines whether to turn on Amazon ECS managed tags for the tasks in the service.

Only tasks launched after the update will reflect the update. To update the tags on all tasks, use the force deployment option.

You can change this parameter for rolling deployments, and blue/green deployments.

Enable ECS Exec

Determines whether Amazon ECS Exec is used.

If you do not want to override the value that was set when the service was created, you can set this to null when performing this action.

You can change this parameter for rolling deployments.

Health check grace period

The period of time, in seconds, that the Amazon ECS service scheduler ignores unhealthy Elastic Load Balancing, VPC Lattice, and container health checks after a task has first started. If you don't specify a health check grace period value, the default value of `0` is used. If you don't use any of the health checks, then `healthCheckGracePeriodSeconds` is unused.

If your service's tasks take a while to start and respond to health checks, you can specify a health check grace period of up to 2,147,483,647 seconds (about 69 years). During that time, the Amazon ECS service scheduler ignores health check status. This grace period can prevent the service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

You can change this parameter for rolling deployments and blue/green deployments.

Load balancers

You must use a service-linked role when you update a load balancer.

A list of Elastic Load Balancing load balancer objects. It contains the load balancer name, the container name, and the container port to access from the load balancer. The container name is as it appears in a container definition.

Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

When you add, update, or remove a load balancer configuration, Amazon ECS starts new tasks with the updated Elastic Load Balancing configuration, and then stops the old tasks when the new tasks are running.

For services that use rolling updates, you can add, update, or remove Elastic Load Balancing target groups. You can update from a single target group to multiple target groups and from multiple target groups to a single target group.

For services that use blue/green deployments, you can update Elastic Load Balancing target groups by using `CreateDeployment` through CodeDeploy. Note that multiple target groups are not supported for blue/green deployments. For more information see Register multiple target groups with a service.

For services that use the external deployment controller, you can add, update, or remove load balancers by using CreateTaskSet. Note that multiple target groups are not supported for external deployments. For more information see Register multiple target groups with a service.

Pass an empty list to remove load balancers.

You can change this parameter for rolling deployments.

Network configuration

The service network configuration.

You can change this parameter for rolling deployments.

Placement constraints

An array of task placement constraint objects to update the service to use. If no value is specified, the existing placement constraints for the service will remain unchanged. If this value is specified, it will override any existing placement constraints defined for the service. To remove all existing placement constraints, specify an empty array.

You can specify a maximum of 10 constraints for each task. This limit includes constraints in the task definition and those specified at runtime.

You can change this parameter for rolling deployments, and blue/green deployments.

Placement strategy

The task placement strategy objects to update the service to use. If no value is specified, the existing placement strategy for the service will remain unchanged. If this value is specified, it will override the existing placement strategy defined for the service. To remove an existing placement strategy, specify an empty object.

You can change this parameter for rolling deployments, and blue/green deployments.

Platform version

The Fargate platform version your service runs on.

A service using a Linux platform version cannot be updated to use a Windows platform version and vice versa.

You can change this parameter for rolling deployments.

Propagate tags

Determines whether to propagate the tags from the task definition or the service to the task. If no value is specified, the tags aren't propagated.

Only tasks launched after the update will reflect the update. To update the tags on all tasks, set `forceNewDeployment` to `true`, so that Amazon ECS starts new tasks with the updated tags.

You can change this parameter for rolling deployments, and blue/green deployments.

Service Connect configuration

The configuration for Amazon ECS Service Connect. This parameter determines how the service connects to other services within your application.

You can change this parameter for rolling deployments.

Service registries

You must use a service-linked role when you update the service registries.

The details for the service discovery registries to assign to this service. For more information, see Service Discovery.

When you add, update, or remove the service registries configuration, Amazon ECS starts new tasks with the updated service registries configuration, and then stops the old tasks when the new tasks are running.

Pass an empty list to remove the service registries.

You can change this parameter for rolling deployments.

Task definition

The task definition and revision to use for the service.

If you change the ports used by containers in a task definition, you might need to update the security groups for the container instances to work with the updated ports.

If you update the task definition for the service, the container name and container port that are specified in the load balancer configuration must remain in the task definition.

The container image pull behavior differs for the launch types. For more information, see one of the following:

- Fargate launch type for Amazon ECS
- EC2 launch type for Amazon ECS
- External (Amazon ECS Anywhere) launch type for Amazon ECS

You can change this parameter for rolling deployments.

Volume configuration

The details of the volume that was `configuredAtLaunch`. When `configuredAtLaunch` is set to `true` in the task definition, this service parameter configures one Amazon EBS volume for each task in the service to be created and attached during deployment. You can configure the size, volumeType, IOPS, throughput, snapshot and encryption in ServiceManagedEBSVolumeConfiguration. The `name` of the volume must match the `name` from the task definition. If set to null, no new deployment is triggered. Otherwise, if this configuration differs from the existing one, it triggers a new deployment.

You can change this parameter for rolling deployments.

VPC Lattice configuration

The VPC Lattice configuration for your service. This defines how your service integrates with VPC Lattice for service-to-service communication.

You can change this parameter for rolling deployments.

AWS CDK considerations

The AWS CDK doesn't track resource states. It doesn't know whether you are creating or updating a service. Customers should use the escape hatch to access the ecs `Service` L1 construct directly.

For information about escape hatches, see [Customize constructs from the AWS Construct Library]() in the *AWS Cloud Development Kit (AWS CDK) v2 Developer Guide*.

To migrate your existing service to the `ecs.Service` construct, do the following:

1. Use the escape hatch to access the `Service` L1 construct.
2. Manually set the following properties in the `Service` L1 construct.

   If your service uses Amazon EC2 capacity:

   - `daemon?`
   - `placementConstraints?`
   - `placementStrategies?`
   - If you use the `awsvpc` network mode, you need to set the `vpcSubnets?` and the `securityGroups?` constructs.

   If your service uses Fargate:

   - `FargatePlatformVersion`
   - The `vpcSubnets?` and the `securityGroups?` constructs.
3. Set the `launchType` as follows:

```
const cfnEcsService = service.node.findChild('Service') as ecs.CfnService;
cfnEcsService.launchType = "FARGATE";
```

To migrate from a launch type to a capacity provider, do the following:

1. Use the escape hatch to access the `Service` L1 construct.
2. Add the `capacityProviderStrategies?` construct.

3. Deploy the service.

Updating an Amazon ECS service

After you create a service, there are times when you might need to update the service parameters, for example the number of tasks.

When you update a service that uses Amazon ECS circuit breaker, Amazon ECS creates a service deployment and a service revision. These resources allow you to view detailed information about the service history. For more information, see View service history using Amazon ECS service deployments.

Prerequisites

Before updating a service, verify which service parameters can be changed for your deployment type. For a complete list of changeable parameters, see Update Amazon ECS service parameters.

Procedure

Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, select the check box next to the service, and then choose **Update**.
4. To have your service start a new deployment, select **Force new deployment**.
5. For **Task definition**, choose the task definition family and revision.

   > **Important**
   >
   > The console validates that the selected task definition family and revision are compatible with the defined compute configuration. If you receive a warning, verify both your task definition compatibility and the compute configuration that you selected.

6. If you chose **Replica**, for **Desired tasks**, enter the number of tasks to launch and maintain in the service.

7. If you chose **Replica**, to have Amazon ECS monitor the distribution of tasks across Availability Zones, and redistribute them when there is an imbalance, under **Availability Zone service rebalancing**, select **Availability Zone service rebalancing**.

8. For **Min running tasks**, enter the lower limit on the number of tasks in the service that must remain in the `RUNNING` state during a deployment, as a percentage of the desired number of tasks (rounded up to the nearest integer). For more information, see [Deployment configuration](#).

9. For **Max running tasks**, enter the upper limit on the number of tasks in the service that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer).

10. To configure how tasks are deployed for your service, expand **Deployment options** and then configure your options.

    a. For **Deployment controller type**, specify the service deployment controller. The Amazon ECS console supports the following controller types: `ECS`.

    b. For **Deployment strategy**, choose the strategy used by Amazon ECS to deploy new versions of the service.

    c. Depending on the choice of **Deployment strategy**, do the following:

Deployment strategy	Steps	
Rolling update	a. For **Min running tasks %**, specify a minimum percentage value of tasks that must run during a service deployment. For more information, see [Deploy Amazon ECS services by replacing tasks](#). b. For **Max running tasks %**, specify a maximum percentage value of tasks that can	

Deployment strategy	Steps	
	run during a service deployment. For more information, see Deploy Amazon ECS services by replacing tasks.	
Blue/green	For **Bake time**, specify a time duration, in minutes, that blue and green service revisions should run simultaenously. For more information, see Amazon ECS blue/green deployments.	

d. To run Lambda functions for a lifecycle stage, under **Deployment lifecyce hooks** do the following for each unique Lambda function:

   i. Choose **Add**.

      Repeat for every unique function you want to run.

   ii. For **Lambda function**, enter the function name.

   iii. For **Role**, choose the role that you created in the prerequisites with the blue/green permissions.

      For more information, see Permissions required for Lambda functions in Amazon ECS blue/green deployments.

   iv. For **Lifecycle stages**, select the stages the Lambda function runs.

   v. (Optional) For **Hook details**, enter a key value pair that provides information about the hook.

11. To configure how Amazon ECS detects and handles deployment failures, expand **Deployment failure detection**, and then choose your options.

   a. To stop a deployment when the tasks cannot start, select **Use the Amazon ECS deployment circuit breaker**.

To have the software automatically roll back the deployment to the last completed deployment state when the deployment circuit breaker sets the deployment to a failed state, select **Rollback on failures**.

b. To stop a deployment based on application metrics, select **Use CloudWatch alarm(s)**. Then, from **CloudWatch alarm name**, choose the alarms. To create a new alarm, go to the CloudWatch console.

   To have the software automatically roll back the deployment to the last completed deployment state when a CloudWatch alarm sets the deployment to a failed state, select **Rollback on failures**.

12. To change the compute options, expand **Compute configuration**, and then do the following:

    a. For services on AWS Fargate, for **Platform version**, choose the new version.

    b. For services that use a capacity provider strategy, for **Capacity provider strategy**, do the following:

       - To add an additional capacity provider, choose **Add more**. Then, for **Capacity provider**, choose the capacity provider.
       - To remove a capacity provider, to the right of the capacity provider, choose **Remove**.

       A service that's using an Auto Scaling group capacity provider can't be updated to use a Fargate capacity provider. A service that's using a Fargate capacity provider can't be updated to use an Auto Scaling group capacity provider.

13. (Optional) To configure service Auto Scaling, expand **Service auto scaling**, and then specify the following parameters.To use predicte auto scaling, which looks at past load data from traffic flows, configure it after you create the service. For more information, see [Use historical patterns to scale Amazon ECS services with predictive scaling](#).

    a. To use service auto scaling, select **Service auto scaling**.

    b. For **Minimum number of tasks**, enter the lower limit of the number of tasks for service auto scaling to use. The desired count will not go below this count.

    c. For **Maximum number of tasks**, enter the upper limit of the number of tasks for service auto scaling to use. The desired count will not go above this count.

d. Choose the policy type. Under **Scaling policy type**, choose one of the following options.

To use this policy type	Do this	
Target tracking	a. For **Scaling policy type**, choose **Target tracking**. b. For **Policy name**, enter the name of the policy. c. For **ECS service metric**, select one of the following metrics. • **ECSServiceAverageCPUUtilization** – Average CPU utilization of the service. • **ECSServiceAverageMemoryUtilization** – Average memory utilization of the service. • **ALBRequestCountPerTarget** – Number of requests completed per target in an Application Load Balancer target group. d. For **Target value**, enter the value the service maintains for the selected metric. e. For **Scale-out cooldown period**, enter the amount of	

To use this policy type	Do this	
	time, in seconds, after a scale-out activity (add tasks) that must pass before another scale-out activity can start. f. For **Scale-in cooldown period**, enter the amount of time, in seconds, after a scale-in activity (remove tasks) that must pass before another scale-in activity can start. g. To prevent the policy from performing a scale-in activity, select **Turn off scale-in**. h. • (Optional) Select **Turn off scale-in** if you want your scaling policy to scale out for increased traffic but don't need it to scale in when traffic decreases.	

To use this policy type	Do this	
Step scaling	a. For **Scaling policy type**, choose **Step scaling**. b. For **Policy name**, enter the policy name. c. For **Alarm name**, enter a unique name for the alarm. d. For **Amazon ECS service metric**, choose the metric to use for the alarm. e. For **Statistic**, choose the alarm statistic. f. For **Period**, choose the period for the alarm. g. For **Alarm condition**, choose how to compare the selected metric to the defined threshold. h. For **Threshold to compare metrics** and **Evaluation period to initiate alarm**, enter the threshold used for the alarm and how long to evaluate the threshold. i. Under **Scaling actions**, do the following: • For **Action**, select whether to add,	

To use this policy type	Do this	
	remove, or set a specific desired count for your service. • If you chose to add or remove tasks, for **Value**, enter the number of tasks (or percent of existing tasks) to add or remove when the scaling action is initiated. If you chose to set the desired count, enter the number of tasks. For **Type**, select whether the **Value** is an integer or a percent value of the existing desired count. • For **Lower bound** and **Upper bound**, enter the lower boundary and upper boundary of your step scaling adjustment. By default, the lower bound for an add policy is the alarm threshold and the upper bound is	

To use this policy type	Do this	
	positive (+) infinity. By default, the upper bound for a remove policy is the alarm threshold and the lower bound is negative (-) infinity. • (Optional) Add additional scaling options. Choose **Add new scaling action**, and then repeat the **Scaling actions** steps. • For **Cooldown period**, enter the amount of time, in seconds, to wait for a previous scaling activity to take effect. For an add policy, this is the time after a scale-out activity that the scaling policy blocks scale-in activities and limits how many tasks can be scale out at a time. For a remove policy, this is the time after a scale-in activity that must pass before	

To use this policy type	Do this
	another scale-in activity can start.

14. (Optional) To use Service Connect, select **Turn on Service Connect**, and then specify the following:

    a. Under **Service Connect configuration**, specify the client mode.

       - If your service runs a network client application that only needs to connect to other services in the namespace, choose **Client side only**.
       - If your service runs a network or web service application and needs to provide endpoints for this service, and connects to other services in the namespace, choose **Client and server**.

    b. To use a namespace that is not the default cluster namespace, for **Namespace**, choose the service namespace.

15. If your task uses a data volume that's compatible with configuration at deployment, you can configure the volume by expanding **Volume**.

    The volume name and volume type are configured when you create a task definition revision and can't be changed when you update a service. To update the volume name and type, you must create a new task definition revision and update the service by using the new revision.

To configure this volume type	Do this
Amazon EBS	a. For **EBS volume type**, choose the type of EBS volume that you want to attach to your task. b. For **Size (GiB)**, enter a valid value for the volume size in gibibytes (GiB). You can specify a minimum of 1 GiB and a

To configure this volume type	Do this	
	maximum of 16,384 GiB volume size. This value is required unless you provide a snapshot ID. c. For **IOPS**, enter the maximum number of input/output operation s (IOPS) that the volume should provide. This value is configurable only for `io1`,`io2`, and `gp3` volume types. d. For **Throughput (MiB/s)**, enter the throughput that the volume should provide, in mebibytes per second (MiBps, or MiB/s). This value is configurable only for the `gp3` volume type. e. For **Snapshot ID**, choose an existing Amazon EBS volume snapshot or enter the ARN of a snapshot if you want to create a volume from a snapshot. You can also create a new, empty volume by not choosing or entering a snapshot ID. f. If you specify a **Snapshot ID**, you	

To configure this volume type	Do this	
	can specify a **Volume initialization rate (MiB/s)**. Enter a value between 100 and 300, in MiB/s, that will determine how fast data is loaded from the snapshot specified using **Snapshot ID** for volume creation. g. For **File system type**, choose the type of file system that will be used for data storage and retrieval on the volume. You can choose either the operating system default or a specific file system type. The default for Linux is XFS. For volumes created from a snapshot, you must specify the same filesystem type that the volume was using when the snapshot was created. If there is a filesystem type mismatch, the task will fail to start. h. For **Infrastructure role**, choose an IAM role with the necessary	

To configure this volume type	Do this	
	permissions that allow Amazon ECS to manage Amazon EBS volumes for tasks. You can attach the `AmazonECSInfrastructureRolePolicyForVolumes` managed policy to the role, or you can use the policy as a guide to create and attach an your own policy with permissions that meet your specific needs. For more information about the necessary permissions, see [Amazon ECS infrastructure IAM role](). i. For **Encryption**, choose **Default** if you want to use the Amazon EBS encryption by default settings. If your account has [Encryption by default]() configured, the volume will be encrypted with the AWS Key Management Service (AWS KMS) key that's specified in the setting. If you choose **Default** and Amazon EBS default	

To configure this volume type	Do this	
	encryption isn't turned on, the volume will be unencrypted. If you choose **Custom**, you can specify an AWS KMS key of your choice for volume encryption. If you choose **None**, the volume will be unencrypted unless you have encryption by default configured, or if you create a volume from an encrypted snapshot. j. If you've chosen **Custom** for **Encryption**, you must specify the AWS KMS key that you want to use. For **KMS key**, choose an AWS KMS key or enter a key ARN. If you choose to encrypt your volume by using a symmetric customer managed key, make sure that you have the right permissions defined in your AWS KMS key policy. For more information, see Data	

To configure this volume type	Do this	
	encryption for Amazon EBS volumes. k. (Optional) Under **Tags**, you can add tags to your Amazon EBS volume by either propagati ng tags from the task definition or service, or by providing your own tags. If you want to propagate tags from the task definition, choose **Task definition** for **Propagate tags from**. If you want to propagate tags from the service, choose **Service** for **Propagate tags from**. If you choose **Do not propagate**, or if you don't choose a value, the tags aren't propagated. If you want to provide your own tags, choose **Add tag** and then provide the key and value for each tag you add. For more information about tagging Amazon EBS volumes, see	

To configure this volume type	Do this	
	Tagging Amazon EBS volumes.	

16. (Optional) To help identify your service, expand the **Tags** section, and then configure your tags.

    - [Add a tag] Choose **Add tag**, and do the following:
      - For **Key**, enter the key name.
      - For **Value**, enter the key value.
    - [Remove a tag] Next to the tag, choose **Remove tag**.
17. Choose **Update**.

AWS CLI

- Run `update-service`. For information about running the command, see update-service in the AWS Command Line Interface Reference.

  The following `update-service` example updates the desired task count of the service `my-http-service` to 2.

  Replace the *`user-input`* with your values.

  ```
  aws ecs update-service \
      --cluster MyCluster \
      --service my-http-service \
      --desired-count 2
  ```

Next steps

Track your deployment and view your service history for services that Amazon ECS circuit breaker. For more information, see View service history using Amazon ECS service deployments.

Deleting an Amazon ECS service using the console

The following are some of the reasons you would delete a service:

- The application is no longer needed
- You are migrating the service to a new environment
- The application is not actively being used
- The application is using more resources than needed and you are trying to optmize your costs

The service is automatically scaled down to zero before it is deleted. Load balancer resources or service discovery resources associated with the service are not affected by the service deletion. To delete your Elastic Load Balancing resources, see one of the following topics, depending on your load balancer type: Delete an Application Load Balancer or Delete a Network Load Balancer.

When you delete a service, Amazon ECS deletes all service deployments and service revisions for the service.

When you delete a service, if there are still running tasks that require cleanup, the service status moves from `ACTIVE` to `DRAINING`, and the service is no longer visible in the console or in the `ListServices` API operation. After all tasks have transitioned to either `STOPPING` or `STOPPED` status, the service status moves from `DRAINING` to `INACTIVE`. Services in the `DRAINING` or `INACTIVE` status can still be viewed with the `DescribeServices` API operation.

> **Important**
>
> If you attempt to create a new service with the same name as an existing service in either `ACTIVE` or `DRAINING` status, you'll receive an error.

Procedure

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, select the cluster for the service.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : *name*** page, choose the **Services** tab.
5. Select the services, and then choose **Delete**.
6. To delete a service even if it wasn't scaled down to zero tasks, select **Force delete service**.
7. At the confirmation prompt, enter **delete**, and then choose **Delete**.

Migrate an Amazon ECS short service ARN to a long ARN

Amazon ECS assigns a unique Amazon Resource Name (ARN) to each service. Services that were created before 2021 have a short ARN format:

```
arn:aws:ecs:region:aws_account_id:service/service-name
```

Amazon ECS changed the ARN format to include the cluster name. This is a long ARN format:

```
arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name
```

Your service must have the long ARN format in order to tag your service.

You can migrate a service with a short ARN format to the long ARN format without having to recreate the service. You can use the API, CLI, or the console. You can't undo the migration operation.

The migration process is seamless and ensures zero downtime for your service. During migration:

- **Service availability**: Your service continues to run normally with no interruption to traffic or functionality.
- **Running tasks**: Existing tasks continue to run without disruption. New tasks launched after migration will use the long ARN format if the `taskLongArnFormat` account setting is enabled.
- **Container instances**: Container instances are not affected by the service ARN migration and continue to operate normally.
- **Service configuration**: All service settings, including task definition, networking, and load balancer configurations, remain unchanged.

If you want to use AWS CloudFormation to tag a service with short ARN format, you must migrate the service using the API, CLI, or console. After the migration completes you can use AWS CloudFormation to tag the service.

If you want to use Terraform to tag a service with short ARN format, you must migrate the service using the API, CLI, or console. After the migration completes you can use Terraform to tag the service.

After the migration is complete, the service has the following changes:

- The long ARN format

```
arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name
```

- When you migrate using the console, Amazon ECS adds a tag to the service with the key set to "ecs:serviceArnMigratedAt" and the value set to the migration timestamp (UTC format).

  This tag counts toward your tag quota.
- When the `PhysicalResourceId` in a AWS CloudFormation stack represents a service ARN, the value does not change and will continue to be the short service ARN.

Prerequisites

Perform the following operations before you migrate the service ARN.

1. To see if you have a short service ARN, view the service details in the Amazon ECS console (you see a warning when the service has the short ARN format), or the `serviceARN` return parameter from `describe-services`. When the ARN does not include the cluster name, you have a short ARN. The following is the format of a short ARN:

   ```
   arn:aws:ecs:region:aws_account_id:service/service-name
   ```
2. Note the created at date.
3. If you have IAM policies that use the short ARN format, update it to the long ARN format.

   Replace each *`user input placeholder`* with your own information.

   ```
   arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name
   ```

   For more information, see Editing IAM policies in the *AWS Identity and Access Management User Guide*.
4. If you have tools that use the short ARN format, update it to the long ARN format.

   Replace each *`user input placeholder`* with your own information.

   ```
   arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name
   ```
5. Enable the service long ARN format. Run `put-account-setting` with the `serviceLongArnFormat` option set to `enabled`. For more information, see, put-account-setting in the *Amazon Elastic Container Service API Reference*.

   Run the command as the root user when your service has an unknown `createdAt` date.

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled
```

Example output

```
{
    "setting": {
        "name": "serviceLongArnFormat",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:role/your-role",
        "type": user
    }
}
```

6. Enable the task long ARN format. This account setting controls the ARN format for new tasks that are launched after the service migration is complete. Run `put-account-setting` with the `taskLongArnFormat` option set to `enabled`. For more information, see, put-account-setting in the *Amazon Elastic Container Service API Reference*.

   Run the command as the root user when your service has an unknown `createdAt` date.

   ```
   aws ecs put-account-setting --name taskLongArnFormat --value enabled
   ```

   Example output

   ```
   {
       "setting": {
           "name": "taskLongArnFormat",
           "value": "enabled",
           "principalArn": "arn:aws:iam::123456789012:role/your-role",
           "type": user
       }
   }
   ```

   Note

   The `taskLongArnFormat` setting does not directly migrate existing tasks. It only affects the ARN format of new tasks that are created after the setting is enabled.

Existing running tasks retain their current ARN format until they are replaced through normal service operations (such as deployments or scaling activities).

Procedure

Use the following to migrate your service ARN.

Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. In the **Services** section, choose a service that has a warning in the ARN column.

   The service details page appears.
4. Choose **Migrate to long ARN**.

   The Migrate service dialog box appears.
5. Choose **Migrate**.

CLI

After you complete the prerequisites, you can tag your service. Run the following command:

Amazon ECS considers passing the long ARN format in a `tag-resource` API request for a service with a short ARN as a signal to migrate the service to use the long ARN format.

```
aws ecs tag-resource \
    --resource-arn arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name
    --tags key=key1,value=value1
```

The following example tags MyService with a tag that has a key set to "TestService" and a value set to "WebServers:

```
aws ecs tag-resource \
    --resource-arn arn:aws:ecs:us-east-1:123456789012:service/MyCluster/MyService
    --tags key=TestService1,value=WebServers
```

Terraform

After you complete the prerequisites, you can tag your service. Create an `aws_ecs_service` resource and set the `tags` reference. For more information, see [Resource: aws_ecs_service](#) in the Terraform documentation.

```
resource "aws_ecs_service" "MyService" {
  name    = "example"
  cluster = aws_ecs_cluster.MyService.id

 tags = {
 "Name"  =  "MyService"
 "Environment"  =  "Production"
 "Department"  =  "QualityAssurance"
  }
}
```

Next steps

You can add tags to the service. For more information, see [Adding tags to Amazon ECS resources](#).

If you want Amazon ECS to propagate the tags from the task definition or the service to the task, run `update-service` with the `propagateTags` parameter. For more information, see [update-service](#) in the *AWS Command Line Interface Reference*.

Troubleshooting

Some users might encounter the following error when they migrate from the short ARN format to the long ARN format.

`There was an error while migrating the ARN of service` *`service-name`*`. The specified account does not have serviceLongArnFormat or taskLongArnFormat account settings enabled. Add account settings in order to enable tagging.`

If you have already enabled the `serviceLongArnFormat` account setting but still encounter this error, it might be because the account settings for the long ARN format hasn't been enabled for the specific IAM principal that originally created the service.

1. Identify the principal that created the service.

    1. In the console, the information is available in the **Created by** field in the **Configuration and networking** tab on the Service details page in the Amazon ECS console.

2. For the AWS CLI, run the following command:

   Replace the *user-input* with your values.

```
aws ecs describe-services --cluster cluster-name --services service-name --query
 'services[0].{createdBy: createdBy}'
```

2. Enable the required account settings for that specific principal. You can do this in one of the following ways:

   a. Assume the IAM user or role for that principal. Then run `put-account-setting`.

   b. Use the root user to run the command while specifying the creating principal with the `principal-arn`.

      Example.

      Replace the *principal-arn* with the value from Step 1.

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --
principal-arn arn:aws:iam::123456789012:role/jdoe
```

Both methods enable the required `serviceLongArnFormat` account setting on the principal that created the service, which allows the ARN migration to proceed.

Amazon ECS service throttle logic

The Amazon ECS service scheduler includes protective logic that throttles task launches when tasks repeatedly fail to start. This helps prevent unnecessary resource consumption and reduces costs.

When tasks in a service fail to transition from `PENDING` to `RUNNING` state and instead move directly to `STOPPED`, the scheduler:

- Incrementally increases the time between restart attempts
- Continues increasing delays up to a maximum of 27 minutes between attempts
- Generates a service event message to notify you of the issue

> **Note**
>
> The maximum delay period of 27 minutes may change in future updates.

When throttling is activated, you receive this service event message:

```
(service service-name) is unable to consistently start tasks successfully.
```

Important characteristics of the throttle logic:

- Services continue retry attempts indefinitely
- The only modification is the increased time between restarts
- There are no user-configurable parameters

Resolving throttling issues

To resolve throttling, you can:

- Update the service to use a new task definition, which immediately returns the service to normal, non-throttled operation. For more information, see Updating an Amazon ECS service.
- Address the underlying cause of the task failures.

Common causes of task failures that trigger throttling include:

- Insufficient cluster resources (ports, memory, or CPU)
  - Indicated by an insufficient resource service event message
- Container image pull failures
  - Can be caused by invalid image names, tags, or insufficient permissions
  - Results in `CannotPullContainerError` in Viewing Amazon ECS stopped task errors
- Insufficient disk space
  - Results in `CannotCreateContainerError` in stopped task errors
  - For resolution steps, see Troubleshoot the Docker `API error (500): devmapper` in Amazon ECS

Important

The following scenarios do NOT trigger throttle logic:

- Tasks that stop after reaching `RUNNING` state
- Tasks stopped due to failed Elastic Load Balancing health checks
- Tasks where the container command exits with a non-zero code after reaching `RUNNING` state

Amazon ECS service definition parameters

A service definition defines how to run your Amazon ECS service. The following parameters can be specified in a service definition.

Launch type

`launchType`

Type: String

Valid values: `EC2` | `FARGATE` | `EXTERNAL`

Required: No

The launch type on which to run your service. If a launch type is not specified, the default `capacityProviderStrategy` is used by default. For more information, see Amazon ECS launch types.

When you update a service, this parameter triggers a new service deployment.

If a `launchType` is specified, the `capacityProviderStrategy` parameter must be omitted.

Capacity provider strategy

`capacityProviderStrategy`

Type: Array of objects

Required: No

The capacity provider strategy to use for the service.

When you update a service, this parameter doesn't trigger a new service deployment.

A capacity provider strategy consists of one or more capacity providers along with the `base` and `weight` to assign to them. A capacity provider must be associated with the cluster to be used in a capacity provider strategy. The PutClusterCapacityProviders API is used to associate a capacity provider with a cluster. Only capacity providers with an `ACTIVE` or `UPDATING` status can be used.

If a `capacityProviderStrategy` is specified, the `launchType` parameter must be omitted. If no `capacityProviderStrategy` or `launchType` is specified, the `defaultCapacityProviderStrategy` for the cluster is used.

If you want to specify a capacity provider that uses an Auto Scaling group, the capacity provider must already be created. New capacity providers can be created with the CreateCapacityProvider API operation.

To use an AWS Fargate capacity provider, specify either the `FARGATE` or `FARGATE_SPOT` capacity providers. The AWS Fargate capacity providers are available to all accounts and only need to be associated with a cluster to be used.

The PutClusterCapacityProviders API operation is used to update the list of available capacity providers for a cluster after the cluster is created.

`capacityProvider`

Type: String

Required: Yes

The short name or full Amazon Resource Name (ARN) of the capacity provider.

`weight`

Type: Integer

Valid range: Integers between 0 and 1,000.

Required: No

The weight value designates the relative percentage of the total number of tasks launched that use the specified capacity provider.

For example, assume that you have a strategy that contains two capacity providers and both have a weight of one. When the base is satisfied, the tasks split evenly across the two capacity providers. Using that same logic, assume that you specify a weight of 1 for *capacityProviderA* and a weight of 4 for *capacityProviderB*. Then, for every one task that is run using *capacityProviderA*, four tasks use *capacityProviderB*.

`base`

Type: Integer

Valid range: Integers between 0 and 100,000.

Required: No

The base value designates how many tasks, at a minimum, to run on the specified capacity provider. Only one capacity provider in a capacity provider strategy can have a base defined.

Task definition

`taskDefinition`

Type: String

Required: No

The `family` and `revision` (`family:revision`) or full Amazon Resource Name (ARN) of the task definition to run in your service. If a `revision` isn't specified, the latest `ACTIVE` revision of the specified family is used.

When you update a service, this parameter triggers a new service deployment.

A task definition must be specified when using the rolling update (`ECS`) deployment controller.

Platform operating system

`platformFamily`

Type: string

Required: Conditional

Default: Linux

This parameter is required for Amazon ECS services hosted on Fargate.

This parameter is ignored for Amazon ECS services hosted on Amazon EC2.

The operating system on the containers that runs the service. The valid values are `LINUX`, `WINDOWS_SERVER_2019_FULL`, `WINDOWS_SERVER_2019_CORE`, `WINDOWS_SERVER_2022_FULL`, and `WINDOWS_SERVER_2022_CORE`.

The `platformFamily` value for every task that you specify for the service must match the service `platformFamily` value. For example, if you set the `platformFamily` to `WINDOWS_SERVER_2019_FULL`, the `platformFamily` value for all the tasks must be `WINDOWS_SERVER_2019_FULL`.

Platform version

`platformVersion`

Type: String

Required: No

The platform version on which your tasks in the service are running. A platform version is only specified for tasks using the Fargate launch type. If one is not specified, the latest version (`LATEST`) is used by default.

When you update a service, this parameter triggers a new service deployment.

AWS Fargate platform versions are used to refer to a specific runtime environment for the Fargate task infrastructure. When specifying the `LATEST` platform version when running a task or creating a service, you get the most current platform version available for your tasks. When you scale up your service, those tasks receive the platform version that was specified on the service's current deployment. For more information, see Fargate platform versions for Amazon ECS.

> **Note**
>
> Platform versions are not specified for tasks using the EC2 launch type.

Cluster

`cluster`

Type: String

Required: No

The short name or full Amazon Resource Name (ARN) of the cluster on which to run your service. If you do not specify a cluster, the `default` cluster is assumed.

Service name

`serviceName`

Type: String

Required: Yes

The name of your service. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a Region or across multiple Regions.

Scheduling strategy

`schedulingStrategy`

Type: String

Valid values: `REPLICA` | `DAEMON`

Required: No

The scheduling strategy to use. If no scheduling strategy is specified, the `REPLICA` strategy is used. For more information, see Amazon ECS services.

There are two service scheduler strategies available:

- `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica scheduling strategy.

- `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see [Daemon scheduling strategy](#).

> **Note**
>
> Fargate tasks do not support the `DAEMON` scheduling strategy.

Desired count

`desiredCount`

Type: Integer

Required: No

The number of instantiations of the specified task definition to place and keep running in your service.

When you update a service, this parameter doesn't trigger a new service deployment.

This parameter is required if the `REPLICA` scheduling strategy is used. If the service uses the `DAEMON` scheduling strategy, this parameter is optional.

When you use service auto scaling, when you update a currently running service with a `desiredCount` less than the number of tasks currently running, the service scales down to the specified `desiredCount`.

Deployment configuration

`deploymentConfiguration`

Type: Object

Required: No

Optional deployment parameters that control how many tasks run during the deployment and the ordering of stopping and starting tasks.

When you update a service, this parameter doesn't trigger a new service deployment.

`maximumPercent`

Type: Integer

Required: No

If a service is using the rolling update (`ECS`) deployment type, the `maximumPercent` parameter represents an upper limit on the number of your service's tasks that are allowed in the `RUNNING`, `STOPPING`, or `PENDING` state during a deployment. It is expressed as a percentage of the `desiredCount` that is rounded down to the nearest integer. You can use this parameter to define the deployment batch size. For example, if your service is using the `REPLICA` service scheduler and has a `desiredCount` of four tasks and a `maximumPercent` value of 200%, the scheduler starts four new tasks before stopping the four older tasks. This is provided that the cluster resources required to do this are available. The default `maximumPercent` value for a service using the `REPLICA` service scheduler is 200%.

The Amazon ECS scheduler uses this parameter to replace unhealthy tasks by starting replacement tasks first and then stopping the unhealthy tasks, as long as cluster resources for starting replacement tasks are available. For more information about how the scheduler replaces unhealthy tasks, see [Amazon ECS services](#).

If your service is using the `DAEMON` service scheduler type, the `maximumPercent` should remain at 100%. This is the default value.

The maximum number of tasks during a deployment is the `desiredCount` multiplied by the `maximumPercent`/100, rounded down to the nearest integer value.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and tasks that use the EC2 launch type, the **maximum percent** value is set to the default value. The value is used to define the upper limit on the number of the tasks in the service that remain in the `RUNNING` state while the container instances are in the `DRAINING` state.

> **Note**
>
> You can't specify a custom `maximumPercent` value for a service that uses either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and has tasks that use the EC2 launch type.

If the service uses either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types, and the tasks in the service use the Fargate launch type, the maximum percent value is not used. The value is still returned when describing your service.

`minimumHealthyPercent`

Type: Integer

Required: No

If a service is using the rolling update (`ECS`) deployment type, the `minimumHealthyPercent` represents a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment. This is expressed as a percentage of the `desiredCount` that is rounded up to the nearest integer. You can use this parameter to deploy without using additional cluster capacity.

For example, if your service has a `desiredCount` of four tasks, a `minimumHealthyPercent` of 50%, and a `maximumPercent` of 100%, the service scheduler stops two existing tasks to free up cluster capacity before starting two new tasks.

If any tasks are unhealthy and if `maximumPercent` doesn't allow the Amazon ECS scheduler to start replacement tasks, the scheduler stops the unhealthy tasks one-by-one — using the `minimumHealthyPercent` as a constraint — to clear up capacity to launch replacement tasks. For more information about how the scheduler replaces unhealthy tasks, see [Amazon ECS services](#).

For services that *do not* use a load balancer, consider the following:

- A service is considered healthy if all essential containers within the tasks in the service pass their health checks.
- If a task has no essential containers with a health check defined, the service scheduler waits for 40 seconds after a task reaches a `RUNNING` state before the task is counted towards the minimum healthy percent total.
- If a task has one or more essential containers with a health check defined, the service scheduler waits for the task to reach a healthy status before counting it towards the minimum healthy percent total. A task is considered healthy when all essential containers within the task have passed their health checks. The amount of time the service scheduler can wait for is determined by the container health check settings. For more information, see [Health check](#).

For services that *do* use a load balancer, consider the following:

- If a task has no essential containers with a health check defined, the service scheduler waits for the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.
- If a task has an essential container with a health check defined, the service scheduler waits for both the task to reach a healthy status and the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.

The default value for a replica service for `minimumHealthyPercent` is 100%. The default `minimumHealthyPercent` value for a service using the `DAEMON` service schedule is 0% for the AWS CLI, the AWS SDKs, and the APIs and 50% for the AWS Management Console.

The minimum number of healthy tasks during a deployment is the `desiredCount` multiplied by the `minimumHealthyPercent`/100, rounded up to the nearest integer value.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and is running tasks that use the EC2 launch type, the **minimum healthy percent** value is set to the default value. The value is used to define the lower limit on the number of the tasks in the service that remain in the `RUNNING` state while the container instances are in the `DRAINING` state.

> **Note**
>
> You can't specify a custom `maximumPercent` value for a service that uses either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and has tasks that use the EC2 launch type.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and is running tasks that use the Fargate launch type, the minimum healthy percent value is not used, although it is returned when describing your service.

Deployment controller

`deploymentController`

Type: Object

Required: No

The deployment controller to use for the service. If no deployment controller is specified, the ECS controller is used. For more information, see Amazon ECS services.

When you update a service, this parameter doesn't trigger a new service deployment.

`type`

Type: String

Valid values: `ECS` | `CODE_DEPLOY` | `EXTERNAL`

Required: yes

The deployment controller type to use. There are three deployment controller types available:

`ECS`

The rolling update (`ECS`) deployment type involves replacing the current running version of the container with the latest version. The number of containers Amazon ECS adds or removes from the service during a rolling update is controlled by adjusting the minimum and maximum number of healthy tasks allowed during a service deployment, as specified in the deploymentConfiguration.

`CODE_DEPLOY`

The blue/green (`CODE_DEPLOY`) deployment type uses the blue/green deployment model powered by CodeDeploy, which allows you to verify a new deployment of a service before sending production traffic to it.

`EXTERNAL`

Use the external deployment type when you want to use any third-party deployment controller for full control over the deployment process for an Amazon ECS service.

Task placement

`placementConstraints`

Type: Array of objects

Required: No

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task. This limit includes constraints in the task definition and those specified at run time. If you use the Fargate launch type, task placement constraints aren't supported.

When you update a service, this parameter doesn't trigger a new service deployment.

`type`

Type: String

Required: No

The type of constraint. Use `distinctInstance` to ensure that each task in a particular group is running on a different container instance. Use `memberOf` to restrict the selection to a group of valid candidates. The value `distinctInstance` is not supported in task definitions.

`expression`

Type: String

Required: No

A cluster query language expression to apply to the constraint. You can't specify an expression if the constraint type is `distinctInstance`. For more information, see [Create expressions to define container instances for Amazon ECS tasks](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cluster-query-language.html).

`placementStrategy`

Type: Array of objects

Required: No

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

When you update a service, this parameter doesn't trigger a new service deployment.

`type`

Type: String

Valid values: `random` | `spread` | `binpack`

Required: No

The type of placement strategy. The `random` placement strategy randomly places tasks on available candidates. The `spread` placement strategy spreads placement across available candidates evenly based on the `field` parameter. The `binpack` strategy places tasks on available candidates that have the least available amount of the resource that's specified with the `field` parameter. For example, if you binpack on memory, a task is placed on the instance with the least amount of remaining memory but still enough to run the task.

`field`

Type: String

Required: No

The field to apply the placement strategy against. For the `spread` placement strategy, valid values are `instanceId` (or `host`, which has the same effect), or any platform or custom attribute that's applied to a container instance, such as `attribute:ecs.availability-zone`. For the `binpack` placement strategy, valid values are `cpu` and `memory`. For the `random` placement strategy, this field is not used.

Tags

`tags`

Type: Array of objects

Required: No

The metadata that you apply to the service to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see Tagging Amazon ECS resources.

When you update a service, this parameter doesn't trigger a new service deployment.

`key`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

`value`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

`enableECSManagedTags`

Type: Boolean

Valid values: `true` | `false`

Required: No

Specifies whether to use Amazon ECS managed tags for the tasks in the service. If no value is specified, the default value is `false`. For more information, see Use tags for billing.

When you update a service, this parameter doesn't trigger a new service deployment.

`propagateTags`

Type: String

Valid values: `TASK_DEFINITION` | `SERVICE`

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation or task creation, use the `TagResource` API action.

When you update a service, this parameter doesn't trigger a new service deployment.

Network configuration

`networkConfiguration`

Type: Object

Required: No

The network configuration for the service. This parameter is required for task definitions that use the `awsvpc` network mode to receive their own Elastic Network Interface, and it isn't supported for other network modes. If using the Fargate launch type, the `awsvpc` network mode is required. For more information about networking for the Amazon EC2 launch type, see Amazon ECS task networking options for the EC2 launch type. For more information about networking for the Fargate launch type, see Amazon ECS task networking options for the Fargate launch type.

When you update a service, this parameter triggers a new service deployment.

`awsvpcConfiguration`

Type: Object

Required: No

An object representing the subnets and security groups for a task or service.

`subnets`

Type: Array of strings

Required: Yes

The subnets that are associated with the task or service. There is a limit of 16 subnets that can be specified according to `awsvpcConfiguration`.

`securityGroups`

Type: Array of strings

Required: No

The security groups associated with the task or service. If you don't specify a security group, the default security group for the VPC is used. There's a limit of five security groups that can be specified based on `awsvpcConfiguration`.

`assignPublicIP`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether the task's elastic network interface receives a public IP address. If no value is specified, the default value of `DISABLED` is used.

`healthCheckGracePeriodSeconds`

Type: Integer

Required: No

The period of time, in seconds, that the Amazon ECS service scheduler ignores unhealthy Elastic Load Balancing, VPC Lattice, and container health checks after a task has first started. If you do not specify a health check grace period value, the default value of 0 is used. If you do not use any of the health checks, then `healthCheckGracePeriodSeconds` is unused.

If your service's tasks take a while to start and respond, you can specify a health check grace period of up to 2,147,483,647 seconds (about 69 years). During that time, the Amazon ECS service scheduler ignores the health check status. This grace period can prevent the service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

When you update a service, this parameter doesn't trigger a new service deployment.

`loadBalancers`

Type: Array of objects

Required: No

A load balancer object representing the load balancers to use with your service. For services that use an Application Load Balancer or Network Load Balancer, there's a limit of five target groups that you can attach to a service.

When you update a service, this parameter triggers a new service deployment.

After you create a service, the load balancer configuration can't be changed from the AWS Management Console. You can use the AWS Copilot, AWS CloudFormation, AWS CLI or SDK to modify the load balancer configuration for the `ECS` rolling deployment controller only, not AWS CodeDeploy blue/green or external. When you add, update, or remove a load balancer configuration, Amazon ECS starts a new deployment with the updated Elastic Load Balancing configuration. This causes tasks to register to and deregister from load balancers. We recommend that you verify this on a test environment before you update the Elastic Load Balancing configuration. For information about how to modify the configuration, see UpdateService in the *Amazon Elastic Container Service API Reference*.

For Application Load Balancers and Network Load Balancers, this object must contain the load balancer target group ARN, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance and port combination is registered as a target in the target group specified.

`targetGroupArn`

Type: String

Required: No

The full Amazon Resource Name (ARN) of the Elastic Load Balancing target group that's associated with a service.

A target group ARN is only specified when using an Application Load Balancer or Network Load Balancer.

`loadBalancerName`

Type: String

Required: No

The name of the load balancer to associate with the service.

If you're using an Application Load Balancer or a Network Load Balancer, omit the load balancer name parameter.

`containerName`

Type: String

Required: No

The name of the container (as it appears in a container definition) to associate with the load balancer.

`containerPort`

Type: Integer

Required: No

The port on the container to associate with the load balancer. This port must correspond to a `containerPort` in the task definition used by tasks in the service. For tasks that use the

EC2 launch type, the container instance must allow inbound traffic on the `hostPort` of the port mapping.

`role`

Type: String

Required: No

The short name or full ARN of the IAM role that allows Amazon ECS to make calls to your load balancer on your behalf. This parameter is only permitted if you are using a load balancer with a single target group for your service, and your task definition does not use the `awsvpc` network mode. If you specify the `role` parameter, you must also specify a load balancer object with the `loadBalancers` parameter.

When you update a service, this parameter doesn't trigger a new service deployment.

If your specified role has a path other than `/`, then you must either specify the full role ARN (this is recommended) or prefix the role name with the path. For example, if a role with the name `bar` has a path of `/foo/` then you would specify `/foo/bar` as the role name. For more information, see [Friendly Names and Paths](#) in the *IAM User Guide*.

> **Important**
>
> If your account has already created the Amazon ECS service-linked role, that role is used by default for your service unless you specify a role here. The service-linked role is required if your task definition uses the awsvpc network mode, in which case you should not specify a role here. For more information, see [Using service-linked roles for Amazon ECS](#).

`serviceConnectConfiguration`

Type: Object

Required: No

The configuration for this service to discover and connect to services, and be discovered by, and connected from, other services within a namespace.

When you update a service, this parameter triggers a new service deployment.

For more information, see Use Service Connect to connect Amazon ECS services with short names.

`enabled`

Type: Boolean

Required: Yes

Specifies whether to use Service Connect with this service.

`namespace`

Type: String

Required: No

The short name or full Amazon Resource Name (ARN) of the AWS Cloud Map namespace for use with Service Connect. The namespace must be in the same AWS Region as the Amazon ECS service and cluster. The type of namespace doesn't affect Service Connect. For more information about AWS Cloud Map, see Working with Services in the *AWS Cloud Map Developer Guide*.

`services`

Type: Array of objects

Required: No

An array of Service Connect service objects. These are names and aliases (also known as endpoints) that are used by other Amazon ECS services to connect to this service.

This field isn't required for a "client" Amazon ECS service that's a member of a namespace only to connect to other services within the namespace. An example is frontend application that accepts incoming requests from either a load balancer that's attached to the service or by other means.

An object selects a port from the task definition, assigns a name for the AWS Cloud Map service, and an array of aliases (also known as endpoints) and ports for client applications to refer to this service.

`portName`

Type: String

Required: Yes

The `portName` must match the `name` of one of the `portMappings` from all of the containers in the task definition of this Amazon ECS service.

`discoveryName`

Type: String

Required: No

The `discoveryName` is the name of the new AWS Cloud Map service that Amazon ECS creates for this Amazon ECS service. This must be unique within the AWS Cloud Map namespace.

If this field isn't specified, `portName` is used.

`clientAliases`

Type: Array of objects

Required: No

The list of client aliases for this service connect service. You use these to assign names that can be used by client applications. The maximum number of client aliases that you can have in this list is 1.

Each alias ("endpoint") is a DNS name and port number that other Amazon ECS services ("clients") can use to connect to this service.

Each name and port combination must be unique within the namespace.

These names are configured within each task of the client service, not in AWS Cloud Map. DNS requests to resolve these names don't leave the task, and don't count toward the quota of DNS requests per second per elastic network interface.

`port`

Type: Integer

Required: Yes

The listening port number for the service connect proxy. This port is available inside of all of the tasks within the same namespace.

To avoid changing your applications in client Amazon ECS services, set this to the same port that the client application uses by default.

dnsName

Type: String

Required: No

The dnsName is the name that you use in the applications of client tasks to connect to this service. The name must be a valid DNS label.

The default value is the discoveryName.namespace if this field is not specified. If the discoveryName isn't specified, the portName from the task definition is used.

To avoid changing your applications in client Amazon ECS services, set this to the same name that the client application uses by default. For example, a few common names are database, db, or the lowercase name of a database, such as mysql or redis.

ingressPortOverride

Type: Integer

Required: No

(Optional) The port number for the Service Connect proxy to listen on.

Use the value of this field to bypass the proxy for traffic on the port number that's specified in the named portMapping in the task definition of this application, and then use it in your Amazon VPC security groups to allow traffic into the proxy for this Amazon ECS service.

In awsvpc mode, the default value is the container port number that's specified in the named portMapping in the task definition of this application. In bridge mode, the default value is the dynamic ephemeral port of the Service Connect proxy.

logConfiguration

Type: LogConfiguration Object

Required: No

This defines where the Service Connect proxy logs are published. Use the logs for debugging during unexpected events. This configuration sets the logConfiguration parameter in the Service Connect proxy container in each task in this Amazon ECS service. The proxy container isn't specified in the task definition.

We recommend that you use the same log configuration as the application containers of the task definition for this Amazon ECS service. For FireLens, this is the log configuration of the application container. It's not the FireLens log router container that uses the `fluent-bit` or `fluentd` container image.

`serviceRegistries`

Type: Array of objects

Required: No

The details of the service discovery configuration for your service. For more information, see [Use service discovery to connect Amazon ECS services with DNS names](#).

When you update a service, this parameter triggers a new service deployment.

`registryArn`

Type: String

Required: No

The Amazon Resource Name (ARN)of the service registry. The currently supported service registry is AWS Cloud Map. For more information, see [Working with Services](#) in the *AWS Cloud Map Developer Guide*.

`port`

Type: Integer

Required: No

The port value that's used if your service discovery service specified an SRV record. This field is required if both the `awsvpc` network mode and SRV records are used.

`containerName`

Type: String

Required: No

The container name value to be used for your service discovery service. This value is specified in the task definition. If the task definition that your service task specifies uses the `bridge` or `host` network mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition that your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

`containerPort`

Type: Integer

Required: No

The port value to be used for your service discovery service. This value is specified in the task definition. If the task definition your service task specifies uses the `bridge` or `host` network mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

Client token

`clientToken`

Type: String

Required: No

The unique, case-sensitive identifier that you provide to ensure the idempotency of the request. It can be up to 32 ASCII characters long.

Availability Zone rebalancing

`availabilityZoneRebalancing`

Type: String

Required: No

Indicates whether the service uses Availability Zone rebalancing. The valid values are `ENABLED` and `DISABLED`.

When you update a service, this parameter doesn't trigger a new service deployment.

Default behavior:

- For new services : If no value is specified, the default is `DISABLED`.
- For existing services : If no value is specified, Amazon ECS defaults the value to to the existing value. If no value was previously set, Amazon ECS sets the value to `DISABLED`.

For more information about Availability Zone rebalancing, see Balancing an Amazon ECS service across Availability Zones.

Volume configurations

`volumeConfigurations`

Type: Object

Required: No

The configuration that will be used to create volumes for tasks that are managed by the service. Only volumes that are marked as `configuredAtLaunch` in the task definition can be configured by using this object.

When you update a service, this parameter triggers a new service deployment.

This object is required for attaching Amazon EBS volumes to tasks that are managed by a service. For more information, see Use Amazon EBS volumes with Amazon ECS.

`name`

Type: String

Required: Yes

The name of a volume that's configured when creating or updating a service. Up to 255 letters (uppercase and lowercase), numbers, underscores (_), and hyphens (-) are allowed. This value must match the volume name that's specified in the task definition.

`managedEBSVolume`

Type: Object

Required: No

The volume configuration used for creating Amazon EBS volumes that are attached to tasks that are maintained by a service when the service is created or updated. One volume is attached per task.

`encrypted`

Type: Boolean

Required: No

Valid values: `true|false`

Specifies whether to encrypt each created Amazon EBS volume. If you've turned on Amazon EBS encryption by default for a particular AWS Region for your AWS account but set this parameter to `false`, this parameter will be overridden, and the volumes will be encrypted with the KMS key specified for encryption by default. For more information about Amazon EBS encryption by default, see Enable Amazon EBS encryption by default in the *Amazon EBS User Guide*. For more information about encrypting Amazon EBS volumes attached to Amazon ECS tasks, see Encrypt data stored in Amazon EBS volumes attached to Amazon ECS tasks.

`kmsKeyId`

Type: String

Required: No

The identifier of the AWS Key Management Service (AWS KMS) key to use for Amazon EBS encryption. If `kmsKeyId` is specified, the encrypted state must be `true`.

The key specified using this parameter overrides the Amazon EBS default or any cluster-level KMS key for Amazon ECS managed storage encryption that you may have specified. For more information, see Encrypt data stored in Amazon EBS volumes attached to Amazon ECS tasks.

You can specify the KMS key by using any of the following:

- **Key ID** – For example, `1234abcd-12ab-34cd-56ef-1234567890ab`.
- **Key alias** – For example, `alias/ExampleAlias`.
- **Key ARN** – For example, `arn:aws:kms:us-east-1:012345678910:key/1234abcd-12ab-34cd-56ef-1234567890ab`.
- **Alias ARN** – For example, `arn:aws:kms:us-east-1:012345678910:alias/ExampleAlias`.

Important

AWS authenticates the KMS key asynchronously. Therefore, if you specify an ID, alias, or ARN that isn't valid, the action can appear to succeed, but it eventually fails. For more information, see Troubleshooting Amazon EBS volume attachment issues.

`volumeType`

Type: String

Required: No

Valid values: `gp2`|`gp3`|`io1`|`io2`|`sc1`|`st1`|`standard`

The Amazon EBS volume type. For more information about volume types, see Amazon EBS volume types in the *Amazon EBS User Guide*. The default volume type is gp3.

Note

The `standard` volume type is not supported for Fargate tasks.

`sizeInGiB`

Type: Integer

Required: No

Valid range: Integers between 1 and 16,384

The size of the EBS volume in gibibytes (GiB). If you do not provide a snapshot ID to configure a volume for attachment, you must provide a size value. If you configure a volume for attachment by using a snapshot, the default value is the snapshot size. You can then specify a size greater than or equal to the snapshot size.

For `gp2` and `gp3` volume types, the valid range is 1-16,384.

For `io1` and `io2` volume types, the valid range is 4-16,384.

For `st1` and `sc1` volume types, the valid range is 125-16,384.

For the `standard` volume type, the valid range is 1-1,024.

`snapshotId`

Type: String

Required: No

The ID of the snapshot of an existing Amazon EBS volume that Amazon ECS uses to create new volumes for attachment. You must specify either a `snapshotId` or a `sizeInGiB`.

`volumeInitializationRate`

Type: Integer

Required: No

The rate, in MiB/s, at which data is fetched from a snapshot of an existing Amazon EBS volume to create new volumes for attachment. This property can be specified only if you specify a `snapshotId`. For more information about this volume initialization rate, including the range of supported rates for initialization, see [Initialize Amazon EBS volumes]() in the *Amazon EBS User Guide*.

`iops`

Type: Integer

Required: No

The number of I/O operations per second (IOPS). For `gp3`, `io1`, and `io2` volumes, this represents the number of IOPS that are provisioned for the volume. For `gp2` volumes, this value represents the baseline performance of the volume and the rate at which the volume accumulates I/O credits for bursting. This parameter is required for `io1` and `io2` volumes. This parameter is not supported for `gp2`, `st1`,`sc1`, or `standard` volumes.

For `gp3` volumes, the valid range of values is 3,000 to 16,000.

For `io1` volumes, the valid range of values is 100 to 64,000.

For `io2` volumes, the valid range of values is 100 to 64,000.

`throughput`

Type: Integer

Required: No

The throughput to provision for the volumes that are attached to tasks that are maintained by a service.

> **Important**
>
> This parameter is supported only for `gp3` volumes.

`roleArn`

Type: String

Required: Yes

The Amazon Resource ARN (ARN) of the infrastructure AWS Identity and Access Management (IAM) role that provides Amazon ECS permissions to manage Amazon EBS resources for your tasks. For more information, see [Amazon ECS infrastructure IAM role](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/infrastructure_IAM_role.html).

`tagSpecifications`

Type: Object

Required: No

The specification for tags to be applied to each Amazon EBS volume.

`resourceType`

Type: String

Required: Yes

Valid values: `volume`

The type of resource to tag on creation.

`tags`

Type: Array of objects

Required: No

The metadata that you apply to volumes to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define.

`AmazonECSCreated` and `AmazonECSManaged` are reserved tags that are added by Amazon ECS on your behalf, so you can specify a maximum of 48 tags of your own. When a volume is deleted, the tags are deleted as well. For more information, see Tagging Amazon ECS resources.

`key`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

One part of a key-value pair that makes up a tag. A key is a general label that acts like a category for more specific tag values.

`value`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

The optional part of a key-value pair that makes up a tag. A value acts as a descriptor within a tag category (key).

`propagateTags`

Type: String

Valid values: `TASK_DEFINITION` | `SERVICE` | `NONE`

Required: No

Specifies whether to copy the tags from the task definition or the service to a volume. If `NONE` is specified or no value is specified, the tags aren't copied.

`fileSystemType`

Type: String

Required: No

Valid values: `xfs`|`ext3`|`ext4`|`NTFS`

The type of file system on a volume. The volume's file system type determines how data is stored and retrieved in the volume. For volumes created from a snapshot, you must specify the same filesystem type that the volume was using when the snapshot was created. If there is a filesystem type mismatch, the task will fail to start.

The valid values for Linux are `xfs`, ext3`, and ext4`. The default for volumes that are attached to Linux tasks is `XFS`.

The valid values for Windows are `NTFS`. The default for volumes that are attached to Windows tasks is `NTFS`.

Service definition template

The following shows the JSON representation of an Amazon ECS service definition.

Amazon EC2 launch type

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
```

```
                "capacityProvider": "",
                "weight": 0,
                "base": 0
            }
        ],
        "platformVersion": "",
        "role": "",
        "deploymentConfiguration": {
            "deploymentCircuitBreaker": {
                "enable": true,
                "rollback": true
            },
            "maximumPercent": 0,
            "minimumHealthyPercent": 0,
            "alarms": {
                "alarmNames": [
                    ""
                ],
                "enable": true,
                "rollback": true
            }
        },
        "placementConstraints": [
            {
                "type": "distinctInstance",
                "expression": ""
            }
        ],
        "placementStrategy": [
            {
                "type": "binpack",
                "field": ""
            }
        ],
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "subnets": [
                    ""
                ],
                "securityGroups": [
                    ""
                ],
                "assignPublicIp": "DISABLED"
            }
```

```
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION",
    "enableExecuteCommand": true,
    "availabilityZoneRebalancing": "ENABLED",
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "",
        "services": [
            {
                "portName": "",
                "discoveryName": "",
                "clientAliases": [
                    {
                        "port": 0,
                        "dnsName": ""
                    }
                ],
                "ingressPortOverride": 0
            }
        ],
        "logConfiguration": {
            "logDriver": "journald",
            "options": {
                "KeyName": ""
            },
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        }
```

```
    },
    "volumeConfigurations": [
        {
            "name": "",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "",
                "volumeType": "",
                "sizeInGiB": 0,
                "snapshotId": "",
                "volumeInitializationRate": 0,
                "iops": 0,
                "throughput": 0,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "",
                                "value": ""
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "",
                "filesystemType": ""
            }
        }
    ]
}
```

Fargate launch type

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
```

```
                "containerPort": 0
            }
        ],
        "serviceRegistries": [
            {
                "registryArn": "",
                "port": 0,
                "containerName": "",
                "containerPort": 0
            }
        ],
        "desiredCount": 0,
        "clientToken": "",
        "launchType": "FARGATE",
        "capacityProviderStrategy": [
            {
                "capacityProvider": "",
                "weight": 0,
                "base": 0
            }
        ],
        "platformVersion": "",
        "platformFamily": "",
        "role": "",
        "deploymentConfiguration": {
            "deploymentCircuitBreaker": {
                "enable": true,
                "rollback": true
            },
            "maximumPercent": 0,
            "minimumHealthyPercent": 0,
            "alarms": {
                "alarmNames": [
                    ""
                ],
                "enable": true,
                "rollback": true
            }
        },
        "placementStrategy": [
            {
                "type": "binpack",
                "field": ""
            }
```

```
    ],
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "EXTERNAL"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "TASK_DEFINITION",
    "enableExecuteCommand": true,
    "availabilityZoneRebalancing": "ENABLED",
    "serviceConnectConfiguration": {
        "enabled": true,
        "namespace": "",
        "services": [
            {
                "portName": "",
                "discoveryName": "",
                "clientAliases": [
                    {
                        "port": 0,
                        "dnsName": ""
                    }
                ],
                "ingressPortOverride": 0
            }
        ],
        "logConfiguration": {
```

```
            "logDriver": "journald",
            "options": {
                "KeyName": ""
            },
            "secretOptions": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ]
        }
    },
    "volumeConfigurations": [
        {
            "name": "",
            "managedEBSVolume": {
                "encrypted": true,
                "kmsKeyId": "",
                "volumeType": "",
                "sizeInGiB": 0,
                "snapshotId": "",
                "volumeInitializationRate": 0,
                "iops": 0,
                "throughput": 0,
                "tagSpecifications": [
                    {
                        "resourceType": "volume",
                        "tags": [
                            {
                                "key": "",
                                "value": ""
                            }
                        ],
                        "propagateTags": "NONE"
                    }
                ],
                "roleArn": "",
                "filesystemType": ""
            }
        }
    ]
}
```

You can create this service definition template using the following AWS CLI command.

```
aws ecs create-service --generate-cli-skeleton
```

Tagging Amazon ECS resources

To help you manage your Amazon ECS resources, you can optionally assign your own metadata to each resource using *tags*. Each *tag* consists of a *key* and an optional *value*.

You can use tags to categorize your Amazon ECS resources in different ways, for example, by purpose, owner, or environment. This is useful when you have many resources of the same type. You can quickly identify a specific resource based on the tags that you assigned to it. For example, you can define a set of tags for your account's Amazon ECS container instances. This helps you track each instance's owner and stack level.

You can use tags for your Cost and Usage reports. You can use these reports to analyze the cost and usage of your Amazon ECS resources. For more information, see the section called “Usage Reports”.

> **Warning**
>
> There are many APIs that return tag keys and their values. Denying access to `DescribeTags` doesn’t automatically deny access to tags returned by other APIs. As a best practice, we recommend that you do not include sensitive data in your tags.

We recommend that you devise a set of tag keys that meets your needs for each resource type. You can use a consistent set of tag keys for easier management of your resources. You can search and filter the resources based on the tags you add.

Tags don't have any semantic meaning to Amazon ECS and are interpreted strictly as a string of characters. You can edit tag keys and values, and you can remove tags from a resource at any time. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the earlier value. When you delete a resource, any tags for the resource are also deleted.

If you use AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to manage tags.

How resources are tagged

There are multiple ways that Amazon ECS tasks, services, task definitions, and clusters are tagged:

- A user manually tags a resource by using the AWS Management Console, Amazon ECS API, the AWS CLI, or an AWS SDK.
- A user creates a service or runs a standalone task and selects the Amazon ECS-managed tags option.

  Amazon ECS automatically tags all newly launched tasks. For more information, see the section called “Amazon ECS-managed tags”.
- A user creates a resource using the console. The console automatically tags the resources.

  These tags are returned in the AWS CLI, and AWS SDK responses and are displayed in the console. You cannot modify or delete these tags.

  For information about the added tags, see the **Tags automatically added by the console** column in the **Tagging support for Amazon ECS resources** table.

If you specify tags when you create a resource and the tags can't be applied, Amazon ECS rolls back the creation process. This ensures that resources are either created with tags or not created at all, and that no resources are left untagged at any time. By tagging resources while they're being created, you can eliminate the need to run custom tagging scripts after resource creation.

The following table describes the Amazon ECS resources that support tagging.

Resource	Supports tags	Supports tag propagation	Tags automatically added by the console
Amazon ECS tasks	Yes	Yes, from the task definition.	*Key*: `aws:ecs:clusterName` *Value*: `cluster-name`
Amazon ECS services	Yes	Yes, from either the task definition or the service to the tasks in the service.	*Key:* `ecs:service:stackId` *Value* `arn:aws:cloudformation:` *`arn`*

Resource	Supports tags	Supports tag propagation	Tags automatically added by the console
Amazon ECS task sets	Yes	No	N/A
Amazon ECS task definitions	Yes	No	*Key*: `ecs:taskDefinition:createdFrom` *Value*: `ecs-console-v2`
Amazon ECS clusters	Yes	No	*Key*: `aws:cloudformation:logical-id` *Value*: `ECSCluster` *Key*: `aws:cloudformation:stack-id` *Value*: `arn:aws:cloudformation: arn` *Key*: `aws:cloudformation:stack-name` *Value*: `ECS-Console-V2-Cluster- EXAMPLE`

Resource	Supports tags	Supports tag propagation	Tags automatically added by the console
Amazon ECS container instances	Yes	Yes, from the Amazon EC2 instance. For more information, see [Adding tags to an Amazon ECS container instance]().	N/A
Amazon ECS External instances	Yes	No	N/A
Amazon ECS capacity provider	Yes. You cannot tag the predefined `FARGATE` and `FARGATE_SPOT` capacity providers.	No	N/A

Tagging resources on creation

The following resources support tagging on creation using the Amazon ECS API, AWS CLI, or AWS SDK:

- Amazon ECS tasks
- Amazon ECS services
- Amazon ECS task definition
- Amazon ECS task sets
- Amazon ECS clusters
- Amazon ECS container instances
- Amazon ECS capacity providers

Amazon ECS has the option to use tagging authorization for resource creation. When the AWS account is configured for tagging authorization, users must have permissions for actions that create the resource, such as `ecsCreateCluster`. If you specify tags in the resource-creating action, AWS performs additional authorization to verify if users or roles have permissions to create tags. Therefore, you must grant explicit permissions to use the `ecs:TagResource` action. For more information, see the section called "Tag resources during creation". For information about how to configure the option, see the section called "Tagging authorization".

Restrictions

The following restrictions apply to tags:

- A maximum of 50 tags can be associated with a resource.
- Tag keys can't be repeated for one resource. Each tag key must be unique, and can only have one value.
- Keys can be up to 128 characters long in UTF-8.
- Values can be up to 256 characters long in UTF-8.
- If multiple AWS services and resources use your tagging schema, limit the types of characters you use. Some services might have restrictions on allowed characters. Generally, allowed characters are letters, numbers, spaces, and the following characters: **+ - = . _ : / @**.
- Tag keys and values are case sensitive.
- You can't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values. These are reserved only for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags-per-resource limit.

Amazon ECS-managed tags

When you use Amazon ECS-managed tags, Amazon ECS automatically tags all newly launched tasks and any Amazon EBS volumes attached to the tasks with the cluster information and either the user-added task definition tags or the service tags. The following describes the added tags:

- Standalone tasks – a tag with a *Key* as `aws:ecs:clusterName` and a *Value* set to the cluster name. All task definition tags that were added by users. An Amazon EBS volume attached to a standalone task will receive the tag with a *Key* as `aws:ecs:clusterName` and a *Value* set to the cluster name. For more information about Amazon EBS volume tagging, see Tagging Amazon EBS volumes.

- Tasks that are part of a service – a tag with a *Key* as `aws:ecs:clusterName` and a *Value* set to the cluster name. A tag with a *Key* as `aws:ecs:serviceName` and a *Value* set to the service name. Tags from one of the following resources:
  - Task definitions – All task definition tags that were added by users.
  - Services – All service tags that were added by users.

    An Amazon EBS volume attached to a task that is part of a service will receive a tag with a *Key* as `aws:ecs:clusterName` and a *Value* set to the cluster name, and a tag with a *Key* as `aws:ecs:serviceName` and a *Value* set to the service name. For more information about Amazon EBS volume tagging, see Tagging Amazon EBS volumes.

The following options are required for this feature:

- You must opt in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see Amazon Resource Names (ARNs) and IDs.
- When you use the APIs to create a service or run a task, you must set `enableECSManagedTags` to `true` for `run-task` and `create-service`. For more information, see create-service and run-task in the *AWS Command Line Interface API Reference*.
- Amazon ECS uses managed tags to determine when some features are enabled, for example cluster Auto Scaling. We recommend that you do not manually modify tags so that the Amazon ECS can effectively manage the features.

Use tags for billing

AWS provides a reporting tool called Cost Explorer that you can use to analyze the cost and usage of your Amazon ECS resources.

You can use Cost Explorer to view charts of your usage and costs. You can view data from the last 13 months, and forecast how much you're likely to spend for the next three months. You can use Cost Explorer to see patterns in how much you spend on AWS resources over time. For example, you can use it to identify areas that need further inquiry and see trends that you can use to understand your costs. You also can specify time ranges for the data, and view time data by day or by month.

You can use Amazon ECS-managed tags or user-added tags for your Cost and Usage Report. For more information, see Amazon ECS usage reports.

To see the cost of your combined resources, you can organize your billing information based on resources that have the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information about setting up a cost allocation report with tags, see The Monthly Cost Allocation Report in the *AWS Billing User Guide*.

Additionally, you can turn on *Split Cost Allocation Data* to get task-level CPU and memory usage data in your Cost and Usage Reports. For more information, see Task-level Cost and Usage Reports.

Note

If you've turned on reporting, it can take up to 24 hours before the data for the current month is available for viewing.

Adding tags to Amazon ECS resources

You can tag new or existing tasks, services, task definitions, or clusters. For information about tagging your container instances, see Adding tags to an Amazon ECS container instance.

Warning

Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags are not intended to be used for private or sensitive data.

You can use the following resources to specify tags when you create the resource.

Task	Console	AWS CLI	API Action
Run one or more tasks.	Running an application as an Amazon ECS task	run-task	RunTask
Create a service.	Creating an Amazon ECS	create-service	CreateService

Task	Console	AWS CLI	API Action
	rolling update deployment		
Create a task set.	Deploy Amazon ECS services using a third-party controller	create-task-set	CreateTaskSet
Register a task definition.	the section called “Creating a task definition using the console”	register-task-definition	RegisterTaskDefinition
Create a cluster.	Creating an Amazon ECS cluster for the Fargate launch type	create-cluster	CreateCluster
Run one or more container instances .	Launching an Amazon ECS Linux container instance	run-instances	RunInstances

Adding tags to existing resources (Amazon ECS console)

You can add or delete tags that are associated with your clusters, services, tasks, and task definitions directly from the resource's page.

To modify a tag for an individual resource

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. From the navigation bar, select the AWS Region to use.
3. In the navigation pane, select a resource type (for example, **Clusters**).
4. Select the resource from the resource list, choose the **Tags** tab, and then choose **Manage tags**.

5. Configure your tags.

   [Add a tag] Choose **Add tag**, and then do the following:

   - For **Key**, enter the key name.
   - For **Value**, enter the key value.

6. Choose **Save**.

Adding tags to existing resources (AWS CLI)

You can add or overwrite one or more tags by using the AWS CLI or an API.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

- AWS CLI - tag-resource
- API action - TagResource

Adding tags to an Amazon ECS container instance

You can associate tags with your container instances using one of the following methods:

- Method 1 – When creating the container instance using the Amazon EC2 API, CLI, or console, specify tags by passing user data to the instance using the container agent configuration parameter `ECS_CONTAINER_INSTANCE_TAGS`. This creates tags that are associated with the container instance in Amazon ECS only, they cannot be listed using the Amazon EC2 API. For more information, see Bootstrapping Amazon ECS Linux container instances to pass data.

  > **Important**
  >
  > If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the ECS_CONTAINER_INSTANCE_TAGS agent configuration parameter to

add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

The following is an example of a user data script that associates tags with your container instance:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- Method 2 – When you create your container instance using the Amazon EC2 API, CLI, or console, first specify tags using the `TagSpecification.N` parameter. Then, pass user data to the instance by using the container agent configuration parameter `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM`. Doing so propagates them from Amazon EC2 to Amazon ECS.

  The following is an example of a user data script that propagates the tags that are associated with an Amazon EC2 instance, and registers the instance with a cluster that's named `MyCluster`.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

  To provide access to allow container instance tags to propagate from Amazon EC2 to Amazon ECS, manually add the following permissions as an inline policy to the Amazon ECS container instance IAM role. For more information, see [Adding and Removing IAM Policies](#).

  - `ec2:DescribeTags`

  The following is an example policy that's used to add these permissions.

  JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
             "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
    ]
}
```

External container instances

You can associate tags with your external container instances by using one of the following methods.

- Method 1 – Before running the installation script to register your external instance with your cluster, create or edit the Amazon ECS container agent configuration file at `/etc/ecs/ecs.config` and add the `ECS_CONTAINER_INSTANCE_TAGS` container agent configuration parameter. This creates tags that are associated with the external instance.

  The following is example syntax.

```
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
```

- Method 2 – After your external instance is registered to your cluster, you can use the AWS Management Console to add tags. For more information, see Adding tags to existing resources (Amazon ECS console).

Amazon ECS usage reports

AWS provides a reporting tool called Cost Explorer that you can use to analyze the cost and usage of your Amazon ECS resources.

You can use Cost Explorer to view charts of your usage and costs. You can view data from the last 13 months, and forecast how much you're likely to spend for the next three months. You can use Cost Explorer to see patterns in how much you spend on AWS resources over time. For example, you can use it to identify areas that need further inquiry and see trends that you can use to understand your costs. You also can specify time ranges for the data, and view time data by day or by month.

The metering data in your Cost and Usage Report shows usage across all of your Amazon ECS tasks. The metering data includes CPU usage as `vCPU-Hours` and memory usage as `GB-Hours` for each task that was run. How that data is presented depends on the launch type of the task.

For tasks using the Fargate launch type, the `lineItem/Operation` column shows `FargateTask` and you will see the cost associated with each task.

For tasks that use the EC2 launch type, the `lineItem/Operation` column shows `ECSTask-EC2` and the tasks don't have a direct cost associated with them. The metering data that's shown in the report, such as memory usage, represents the total resources that the task reserved over the billing period that you specify. You can use this data to determine the cost of your underlying cluster of Amazon EC2 instances. The cost and usage data for your Amazon EC2 instances are listed separately under the Amazon EC2 service.

You can also use the Amazon ECS managed tags to identify the service or cluster that each task belongs to. For more information, see Use tags for billing.

Important

The metering data is only viewable for tasks that are launched on or after November 16, 2018. Tasks that are launched before this date don't show metering data.

The following is an example of some of the fields that you can use to sort cost allocation data in Cost Explorer.

- Cluster name
- Service name
- Resource tags
- Launch type
- AWS Region
- Usage type

For more information about creating an AWS Cost and Usage Report, see AWS Cost and Usage Report in the *AWS Billing User Guide*.

Task-level Cost and Usage Reports

AWS Cost Management can provide CPU and memory usage data in the AWS Cost and Usage Report for the each task on Amazon ECS, including tasks on Fargate and tasks on EC2. This data is called *Split Cost Allocation Data*. You can use this data to analyze costs and usage for applications. Additionally, you can split and allocate the costs to individual business units and teams with cost allocation tags and cost categories. For more information about *Split Cost Allocation Data*, see Understanding split cost allocation data in the AWS Cost and Usage Report User Guide.

You can opt in to task-level *Split Cost Allocation Data* for the account in the AWS Cost Management Console. If you have a management (payer) account, you can opt in from the payer account to apply this configuration to every linked account.

After you set up *Split Cost Allocation Data*, there will be additional columns under the **splitLineItem** header in the report. For more information see Split line item details in the AWS Cost and Usage Report User Guide

For tasks on EC2, this data splits the cost of the EC2 instance based on the resource usage or reservations and the remaining resources on the instance.

The following are prerequistites:

- Set the `ECS_DISABLE_METRICS` Amazon ECS agent configuration parameter to `false`.

  When this setting is `false`, the Amazon ECS agent sends metrics to Amazon CloudWatch. On Linux, this setting is `false` by default and metrics are sent to CloudWatch. On Windows, this setting is `true` by default, so you must change the setting to `false` to send the metrics to CloudWatch for AWS Cost Management to use. For more information about ECS agent configuration, see Amazon ECS container agent configuration.
- The minimum Docker version for reliable metrics is Docker version v20.10.13 and newer, which is included in Amazon ECS-optimized AMI 20220607 and newer.

To use *Split Cost Allocation Data*, you must create a report, and select **Split cost allocation data**. For more information, see Creating Cost and Usage Reports in the AWS Cost and Usage Report User Guide.

AWS Cost Management calculates the *Split Cost Allocation Data* with the task CPU and memory usage. AWS Cost Management can use the task CPU and memory reservation instead of the usage, if the usage is unavailable. If you see the CUR is using the reservations, check that your container instances meet the prerequisites and the task resource usage metrics appear in CloudWatch.

Monitoring Amazon ECS

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon ECS and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. Before you start monitoring Amazon ECS, create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

The metrics made available depend on the launch type of the tasks and services in your clusters. If you are using the Fargate launch type for your services, then CPU and memory utilization metrics are provided to assist in the monitoring of your services. For the Amazon EC2 launch type, you own and need to monitor the EC2 instances that make your underlying infrastructure. Additional CPU and memory reservation and utilization metrics are made available at the cluster, service, and task.

The next step is to establish a baseline for normal Amazon ECS performance in your environment, by measuring performance at various times and under different load conditions. As you monitor Amazon ECS, store historical monitoring data so that you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

To establish a baseline you should, at a minimum, monitor the following items:

- The CPU and memory reservation and utilization metrics for your Amazon ECS clusters
- The CPU and memory utilization metrics for your Amazon ECS services

  For more information, see Viewing Amazon ECS metrics.

Best practices for monitoring Amazon ECS

Use the following best practices for monitoring Amazon ECS.

- Make monitoring a priority to head off small problems before they become big ones
- Create a monitoring plan that includes answers to the following question
  - What are your monitoring goals?
  - What resources will you monitor?
  - How often will you monitor these resources?
  - What monitoring tools will you use?
  - Who will perform the monitoring tasks?
  - Who should be notified when something goes wrong?
- Automate monitoring as much as possible.
- Check the Amazon ECS log files. For more information, see Viewing Amazon ECS container agent logs.
- Use Runtime Monitoring to help protect your accounts, containers, workloads, and the data within your AWS environment. For more information, see Identify unauthorized behavior using Runtime Monitoring.

Monitoring tools for Amazon ECS

AWS provides various tools that you can use to monitor Amazon ECS. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated monitoring tools

You can use the following automated monitoring tools to watch Amazon ECS and report when something is wrong:

- Amazon CloudWatch alarms – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Monitor Amazon ECS using CloudWatch.

For services with tasks that use the Fargate launch type, you can use CloudWatch alarms to scale in and scale out the tasks in your service based on CloudWatch metrics, such as CPU and memory utilization. For more information, see Automatically scale your Amazon ECS service.

For clusters with tasks or services using the EC2 launch type, you can use CloudWatch alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation.

For your container instances that were launched with the Amazon ECS-optimized Amazon Linux AMI, you can use CloudWatch Logs to view different logs from your container instances in one convenient location. You must install the CloudWatch agent on your container instances. For more information, see Download and configure the CloudWatch agent using the command line in the *Amazon CloudWatch User Guide*. You must also add the `ECS-CloudWatchLogs` policy to the `ecsInstanceRole` role. For more information, see Monitoring container instances permissions.

- Amazon CloudWatch Logs – Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the `awslogs` log driver in your task definitions. For more information, see Send Amazon ECS logs to CloudWatch .

  You can also monitor, store, and access the operating system and Amazon ECS container agent log files from your Amazon ECS container instances. This method for accessing logs can be used for containers using the EC2 launch type.

- Amazon CloudWatch Events – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see Automate responses to Amazon ECS errors using EventBridge in this guide and EventBridge is the evolution of Amazon CloudWatch Events in the *Amazon EventBridge User Guide*.

- Container Insights – Collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. Container Insights collects data as performance log events using embedded metric format. These performance log events are entries that use a structured JSON schema that allow high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates aggregated metrics at the cluster, task, and service level as CloudWatch metrics. The metrics that Container Insights collects are available in CloudWatch automatic dashboards, and are also viewable in the Metrics section of the CloudWatch console.

- AWS CloudTrail log monitoring – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see Log Amazon ECS API calls using AWS CloudTrail in this guide, and Working with CloudTrail Log Files in the *AWS CloudTrail User Guide*.
- Runtime Monitoring – Detect threats for clusters and containers within your AWS environment. Runtime Monitoring uses a GuardDuty security agent that adds runtime visibility into individual Amazon ECS workloads, for example, file access, process execution, and network connections.

Manual monitoring tools

Another important part of monitoring Amazon ECS involves manually monitoring those items that the CloudWatch alarms don't cover. The CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your container instances and the containers in your tasks.

- Amazon ECS console:
  - Cluster metrics for the EC2 launch type
  - Service metrics
  - Service health status
  - Service deployment events
- CloudWatch home page:
  - Current alarms and status
  - Graphs of alarms and resources
  - Service health status

  In addition, you can use CloudWatch to do the following:

  - Create customized dashboards to monitor the services you care about.
  - Graph metric data to troubleshoot issues and discover trends.
  - Search and browse all your AWS resource metrics.
  - Create and edit alarms to be notified of problems.
- Container health check - These are commands that run locally on a container and validate application health and availability. You configure these per container in your task definition.

- AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see AWS Trusted Advisor.

  Trusted Advisor has these checks that relate to Amazon ECS:
  - A fault tolerance which indicates that you have a service running in a single Availability Zone.
  - A fault tolerance which indicates that you have not used the spread placement strategy for multiple Availability Zones.
- AWS Compute Optimizer is a service that analyzes the configuration and utilization metrics of your AWS resources. It reports whether your resources are optimal, and generates optimization recommendations to reduce the cost and improve the performance of your workloads.

  For more information, see AWS Compute Optimizer recommendations for Amazon ECS.

Monitor Amazon ECS using CloudWatch

You can monitor your Amazon ECS resources using Amazon CloudWatch, which collects and processes raw data from Amazon ECS into readable, near real-time metrics. These statistics are recorded for a period of two weeks so that you can access historical information and gain a better perspective on how your clusters or services are performing. Amazon ECS metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the Amazon CloudWatch User Guide.

Amazon ECS provides free metrics for clusters and services. For an additional cost, you can turn on Amazon ECS CloudWatch Container Insights for your cluster for per-task metrics, including CPU, memory, and EBS filesystem utilization. For more information about Container Insights, see Monitor Amazon ECS containers using Container Insights with enhanced observability.

Considerations

The following should be considered when using Amazon ECS CloudWatch metrics.

- Any Amazon ECS service hosted on Fargate has CloudWatch CPU and memory utilization metrics automatically, so you don't need to take any manual steps.

- For any Amazon ECS task or service hosted on Amazon EC2 instances, the Amazon EC2 instance requires version `1.4.0` or later (Linux) or `1.0.0` or later (Windows) of the container agent for

CloudWatch metrics to be generated. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS container agent.

- The minimum Docker version for reliable CloudWatch metrics is Docker version `20.10.13` and newer.
- Your Amazon EC2 instances also require the `ecs:StartTelemetrySession` permission on the IAM role that you launch your Amazon EC2 instances with. If you created your Amazon ECS container instance IAM role before CloudWatch metrics were available for Amazon ECS, you might need to add this permission. For information about the container instance IAM role and attaching the managed IAM policy for container instances, see Amazon ECS container instance IAM role.
- You can disable CloudWatch metrics collection on your Amazon EC2 instances by setting `ECS_DISABLE_METRICS=true` in your Amazon ECS container agent configuration. For more information, see Amazon ECS container agent configuration.

Recommended metrics

Amazon ECS provides free CloudWatch metrics you can use to monitor your resources. The CPU and memory reservation and the CPU, memory, and EBS filesystem utilization across your cluster as a whole, and the CPU, memory, and EBS filesystem utilization on the services in your clusters can be measured using these metrics. For your GPU workloads, you can measure your GPU reservation across your cluster.

The infrastructure your Amazon ECS tasks are hosted on in your clusters determines which metrics are available. For tasks hosted on Fargate infrastructure, Amazon ECS provides CPU, memory, and EBS filesystem utilization metrics to assist in the monitoring of your services. For tasks hosted on EC2 instances, Amazon ECS provides CPU, memory, and GPU reservation metrics and CPU and memory utilization metrics at the cluster and service level. You need to monitor the Amazon EC2 instances that make your underlying infrastructure separately. For more information about monitoring your Amazon EC2 instances, see Monitoring Amazon EC2 in the *Amazon EC2 User Guide*.

For information about the recommended alarms to use with Amazon ECS, see one of the following in the *Amazon CloudWatch Logs User Guide*:

- Amazon ECS
- Amazon ECS with Container Insights

Viewing Amazon ECS metrics

After you have resources running in your cluster, you can view the metrics on the Amazon ECS and CloudWatch consoles. The Amazon ECS console provides a 24-hour maximum, minimum, and average view of your cluster and service metrics. The CloudWatch console provides a fine-grained and customizable display of your resources, as well as the number of running tasks in a service.

Amazon ECS console

Amazon ECS service CPU and memory utilization metrics are available on the Amazon ECS console. The view provided for service metrics shows the average, minimum, and maximum values for the previous 24-hour period, with data points available in 5-minute intervals. For more information, see Amazon ECS service utilization metrics.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. Select the cluster that you want to view metrics for.
3. Determine the metrics to view.

To view metrics from	Steps	
Clusters	On the cluster details page, choose the **Metrics** tab. There is also a link provided to the CloudWatch console to view your CloudWatch Container Insights metrics if you have those turned on.	
Services	On the cluster details page, on the **Services** tab, select the service. The metrics are then available on the **Health and metrics** tab.	

CloudWatch console

For the Fargate launch type, Amazon ECS service metrics can also be viewed on the CloudWatch console. The console provides the most detailed view of Amazon ECS metrics, and you can tailor the views to suit your needs. You can view the service utilization and service RUNNING task count.

For the EC2 launch type, Amazon ECS cluster and service metrics can also be viewed on the CloudWatch console. The console provides the most detailed view of Amazon ECS metrics, and you can tailor the views to suit your needs.

For information about how to view the metrics, see View available metrics the *Amazon CloudWatch User Guide*.

Amazon ECS CloudWatch metrics

You can use CloudWatch usage metrics to provide visibility into your accounts usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

Amazon ECS sends metrics to CloudWatch at one-minute intervals. These metrics are collected for resources that have tasks in the `RUNNING` state. If a cluster, service, or other resource has no running tasks, no metrics will be reported for that resource during that period. For example, if you have a cluster with one service but that service has no tasks in a `RUNNING` state, there will be no metrics sent to CloudWatch. Similarly, if you have two services and one of them has running tasks while the other doesn't, only the metrics for the service with running tasks would be sent.

Metric	Description	Valid Dimension	Useful Statistics	Unit
CPUReservation	The percentage of CPU units that are reserved in the cluster or service. The CPU reservation (filtered by `ClusterName`) is measured as the total CPU	`ClusterName` .	Average, Minimum, Maximum	Percent

Metric	Description	Valid Dimension	Useful Statistics	Unit
	units that are reserved by Amazon ECS tasks on the cluster, divided by the total CPU units for all of the Amazon EC2 instances registered in the cluster. Only Amazon EC2 instances in `ACTIVE` or `DRAINING` status will affect CPU reservation metrics. The metric is only supported for tasks hosted on an Amazon EC2 instance.			

Metric	Description	Valid Dimension	Useful Statistics	Unit
`CPUUtilization`	The percentage of CPU units that is used by the cluster or service. The cluster-level CPU utilization (filtered by `ClusterName`) is measured as the total CPU units that are in use by Amazon ECS tasks on the cluster, divided by the total CPU units for all of the Amazon EC2 instances registered in the cluster. Only Amazon EC2 instances in `ACTIVE` or `DRAINING` status will affect CPU reservation metrics. The cluster-level metric is only supported for tasks hosted on an Amazon EC2 instance.	`ClusterName` , `ServiceName`	Average, Minimum, Maximum	Percent

Metric	Description	Valid Dimension	Useful Statistics	Unit
	The service-l evel CPU utilizati on (filtered by `ClusterName` , `ServiceName`) is measured as the total CPU units in use by the tasks that belong to the service, divided by the total number of CPU units that are reserved for the tasks that belong to the service. The service-l evel metric is supported for tasks hosted on Amazon EC2 instances and Fargate.			

Metric	Description	Valid Dimension	Useful Statistics	Unit
`MemoryReservation`	The percentage of memory that is reserved by running tasks in the cluster. Cluster memory reservation is measured as the total memory that is reserved by Amazon ECS tasks on the cluster, divided by the total amount of memory for all of the Amazon EC2 instances registered in the cluster. This metric can only be filtered by `ClusterName` . Only Amazon EC2 instances in `ACTIVE` or `DRAINING` status will affect memory reservati on metrics. The cluster level memory reservation	`ClusterName` .	Average, Minimum, Maximum	Percent

Metric	Description	Valid Dimension	Useful Statistics	Unit
	metric is only supported for tasks hosted on an Amazon EC2 instance. **Note** When calculating memory utilization, if `MemoryReservation` is specified, it's used in the calculation instead of total memory.			

Metric	Description	Valid Dimension	Useful Statistics	Unit
`MemoryUtilization`	The percentage of memory in use by the cluster or service. The cluster-level memory utilization (filtered by `ClusterName`) is measured as the total memory in use by Amazon ECS tasks on the cluster, divided by the total memory for all of the Amazon EC2 instances registered in the cluster. Only Amazon EC2 instances in `ACTIVE` or `DRAINING` status will affect memory utilization metrics. The cluster-level metric is only supported for tasks hosted on an Amazon EC2 instance.	`ClusterName` , `ServiceName`	Average, Minimum, Maximum	Percent

Metric	Description	Valid Dimension	Useful Statistics	Unit
	The service-level memory utilization (filtered by `ClusterName` , `ServiceName`) is measured as the total memory in use by the tasks that belong to the service, divided by the total memory reserved for the tasks that belong to the service. The service-level metric is supported for tasks hosted on Amazon EC2 instances and Fargate.			

Metric	Description	Valid Dimension	Useful Statistics	Unit
`EBSFilesystemUtilization`	The percentage of the Amazon EBS filesystem that is used by tasks in a service. The service level EBS filesystem utilization metric (filtered by `ClusterName` , `ServiceName`) is measured as the total amount of the EBS filesystem in use by the tasks that belong to the service, divided by the total amount of EBS filesystem storage that is allocated for all tasks that belong to the service. The service level EBS filesystem utilization metric is only available for	`ClusterName` , `ServiceName`	Average, Minimum, Maximum	Percent

Metric	Description	Valid Dimension	Useful Statistics	Unit
	tasks hosted on Amazon EC2 instances (using container agent version `1.79.0`) and Fargate (using platform version `1.4.0`) that have an EBS volume attached. **Note** For tasks hosted on Fargate, there is space on the disk that is only used by Fargate. There is no cost associated with the space Fargate uses, but you will see this			

Metric	Description	Valid Dimension	Useful Statistics	Unit
	additional storage using tools like `df`.			

Metric	Description	Valid Dimension	Useful Statistics	Unit
`GPUReservation`	The percentage of total available GPUs that are reserved by running tasks in the cluster. The cluster level GPU reservation metric is measured as the number of GPUs reserved by Amazon ECS tasks on the cluster, divided by the total number of GPUs that was available on all of the Amazon EC2 instances with GPUs registered in the cluster. Only Amazon EC2 instances in `ACTIVE` or `DRAINING` status will affect GPU reservation metrics.	`ClusterName`	Average, Minimum, Maximum	Percent

Metric	Description	Valid Dimension	Useful Statistics	Unit
`ActiveConnectionCount`	The total number of concurrent connections active from clients to the Amazon ECS Service Connect proxies that run in tasks that share the selected `DiscoveryName` . This metric is only available if you have configured Amazon ECS Service Connect. Valid dimensions: `DiscoveryName` and `DiscoveryName`, `ServiceName`, `ClusterName` .	`DiscoveryName` and `DiscoveryName` , `ServiceName` , `ClusterName`	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
`NewConnectionCount`	The total number of new connections established from clients to the Amazon ECS Service Connect proxies that run in tasks that share the selected `DiscoveryName` . This metric is only available if you have configured Amazon ECS Service Connect.	`DiscoveryName` and `DiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Count
`ProcessedBytes`	The total number of bytes of inbound traffic processed by the Service Connect proxies. This metric is only available if you have configured Amazon ECS Service Connect.	`DiscoveryName` and `DiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Bytes

Metric	Description	Valid Dimension	Useful Statistics	Unit
`RequestCount`	The number of inbound traffic requests processed by the Service Connect proxies. This metric is only available if you have configured Amazon ECS Service Connect. You also need to configure `appProtocol` in the port mapping in your task definition.	`DiscoveryName` and `DiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
`GrpcRequestCount`	The number of gRPC inbound traffic requests processed by the Service Connect proxies. This metric is only available if you have configured Amazon ECS Service Connect and the `appProtocol` is GRPC in the port mapping in the task definition.	`DiscoveryName` and `DiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
HTTPCode_Target_2XX_Count	The number of HTTP response codes with numbers 200 to 299 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly. This metric is only available if you have configured Amazon ECS Service Connect and the `appProtocol` is HTTP or HTTP2 in the port mapping in the task definition.	`TargetDiscoveryName` and `TargetDiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
	Valid dimension s:.			

Metric	Description	Valid Dimension	Useful Statistics	Unit
`HTTPCode_Target_3XX_Count`	The number of HTTP response codes with numbers 300 to 399 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly. This metric is only available if you have configured Amazon ECS Service Connect and the `appProtocol` is HTTP or HTTP2 in the port mapping in the task definition.	`TargetDiscoveryName` and `TargetDiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
HTTPCode_Target_4XX_Count	The number of HTTP response codes with numbers 400 to 499 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly. This metric is only available if you have configured Amazon ECS Service Connect and the appProtocol is HTTP or HTTP2 in the port mapping in the task definition.	TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
`HTTPCode_Target_5XX_Count`	The number of HTTP response codes with numbers 500 to 599 generated by the applications in these tasks. These tasks are the targets. This metric only counts the responses sent to the Service Connect proxies by the applications in these tasks, not responses sent directly. This metric is only available if you have configured Amazon ECS Service Connect and the `appProtocol` is HTTP or HTTP2 in the port mapping in the task definition.	`TargetDiscoveryName` and `TargetDiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
RequestCountPerTarget	The average number of requests received by each target that share the selected DiscoveryName . This metric is only available if you have configured Amazon ECS Service Connect.	TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName	Average	Count
TargetProcessedBytes	The total number of bytes processed by the Service Connect proxies. This metric is only available if you have configured Amazon ECS Service Connect.	TargetDiscoveryName and TargetDiscoveryName, ServiceName, ClusterName	Average, Minimum, Maximum, Sum	Bytes

Metric	Description	Valid Dimension	Useful Statistics	Unit
`TargetResponseTime`	The latency of the application request processing. The time elapsed, in milliseconds, after the request reached the Service Connect proxy in the target task until a response from the target application is received back to the proxy. This metric is only available if you have configured Amazon ECS Service Connect.	`TargetDiscoveryName` and `TargetDiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum	Milliseconds

Metric	Description	Valid Dimension	Useful Statistics	Unit
`ClientTLSNegotiationErrorCount`	The total number of times the TLS connection failed. This metric is only used when TLS is enabled. This metric is only available if you have configured Amazon ECS Service Connect.	`DiscoveryName` and `DiscoveryName`, `ServiceName`, `ClusterName`	Average, Minimum, Maximum, Sum	Count

Metric	Description	Valid Dimension	Useful Statistics	Unit
`TargetTLSNegotiationErrorCount`	The total number of times the TLS connection failed due to missing client certificates, failed AWS Private CA verifications, or failed SAN verifications. This metric is only used when TLS is enabled. This metric is only available if you have configured Amazon ECS Service Connect.	`ServiceName` , `ClusterName` , `TargetDiscoveryName` and `TargetDiscoveryName`	Average, Minimum, Maximum, Sum	Count

Dimensions for Amazon ECS metrics

Amazon ECS metrics use the `AWS/ECS` namespace and provide metrics for the following dimensions. Amazon ECS only sends metrics for resources that have tasks in the `RUNNING` state. For example, if you have a cluster with one service in it but that service has no tasks in a `RUNNING` state, there will be no metrics sent to CloudWatch. If you have two services and one of them has running tasks and the other doesn't, only the metrics for the service with running tasks would be sent.

Dimension	Definition
`ClusterName`	This dimension filters the data that you request for all resources in a specified cluster. All Amazon ECS metrics are filtered by `ClusterName` .
`ServiceName`	This dimension filters the data that you request for all resources in a specified service within a specified cluster.
`DiscoveryName`	This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name across all Amazon ECS clusters. Note that a specific port in a running container can have multiple discovery names.
`DiscoveryName, ServiceName, ClusterName`	This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name across tasks that have this discovery name and that are created by this service in this cluster. Use this dimension to see the inbound traffic metrics for a specific service, if you have reused the same discovery name in multiple services in different namespaces. Note that a specific port in a running container can have multiple discovery names.
`TargetDiscoveryName`	This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name across all Amazon ECS clusters.

Dimension	Definition
	Different from `DiscoveryName` , these traffic metrics only measure inbound traffic to this `DiscoveryName` that come from other Amazon ECS tasks that have a Service Connect configuration in this namespace. This includes tasks made by services with either a client-on ly or client-server Service Connect configura tion. Note that a specific port in a running container can have multiple discovery names.
`TargetDiscoveryName, ServiceNa me, ClusterName`	This dimension filters the data that you request for traffic metrics to a specified Service Connect discovery name but only counts traffic from tasks created by this service in this cluster. Use this dimension to see the inbound traffic metrics that come from a specific client in another service. Different from `DiscoveryName, ServiceName, ClusterName` , these traffic metrics only measure inbound traffic to this `DiscoveryName` that come from other Amazon ECS tasks that have a Service Connect configuration in this namespace. This includes tasks made by services with either a client-on ly or client-server Service Connect configura tion. Note that a specific port in a running container can have multiple discovery names.

AWS Fargate usage metrics

You can use CloudWatch usage metrics to provide visibility into your accounts usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

AWS Fargate usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about Fargate service quotas, Amazon ECS endpoints and quotas in the *Amazon Web Services General Reference*..

AWS Fargate publishes the following metrics in the `AWS/Usage` namespace.

Metric	Description
`ResourceCount`	The total number of the specified resource running on your account. The resource is defined by the dimensions associated with the metric.

The following dimensions are used to refine the usage metrics that are published by AWS Fargate.

Dimension	Description
`Service`	The name of the AWS service containing the resource. For AWS Fargate usage metrics, the value for this dimension is `Fargate`.
`Type`	The type of entity that is being reported. Currently, the only valid value for AWS Fargate usage metrics is `Resource`.
`Resource`	The type of resource that is running. The type of resource that is running. Currently, the only valid value for AWS Fargate usage metrics is vCPU which returns information about the running instances.
`Class`	The class of resource being tracked. The class of resource being tracked. For AWS Fargate usage metrics with vCPU as the value of the Resource dimension, the valid values are `Standard/OnDemand` and `Standard/Spot` .

You can use the Service Quotas console to visualize your usage on a graph and configure alarms that alert you when your AWS Fargate usage approaches a service quota. For information about how to create a CloudWatch alarm to notify you when you're close to a quota value threshold, see Service Quotas and Amazon CloudWatch alarms in the *Service Quotas User Guide*

.

Amazon ECS cluster reservation metrics

Cluster reservation metrics are measured as the percentage of CPU, memory, and GPUs that are reserved by all Amazon ECS tasks on a cluster when compared to the aggregate CPU, memory, and GPUs that were registered for each active container instance in the cluster. Only container instances in `ACTIVE` or `DRAINING` status will affect cluster reservation metrics. This metric is used only on clusters with tasks or services hosted on EC2 instances. It's not supported on clusters with tasks hosted on AWS Fargate.

```
                              (Total CPU units reserved by tasks in cluster) x 100
Cluster CPU reservation = 
 --------------------------------------------------------------
                          (Total CPU units registered by container instances in
 cluster)
```

```
                                  (Total MiB of memory reserved by tasks in cluster x
 100)
Cluster memory reservation = 
 ------------------------------------------------------------------
                             (Total MiB of memory registered by container instances in
 cluster)
```

```
                                 (Total GPUs reserved by tasks in cluster x 100)
Cluster GPU reservation = 
 ------------------------------------------------------------------
                             (Total GPUs registered by container instances in cluster)
```

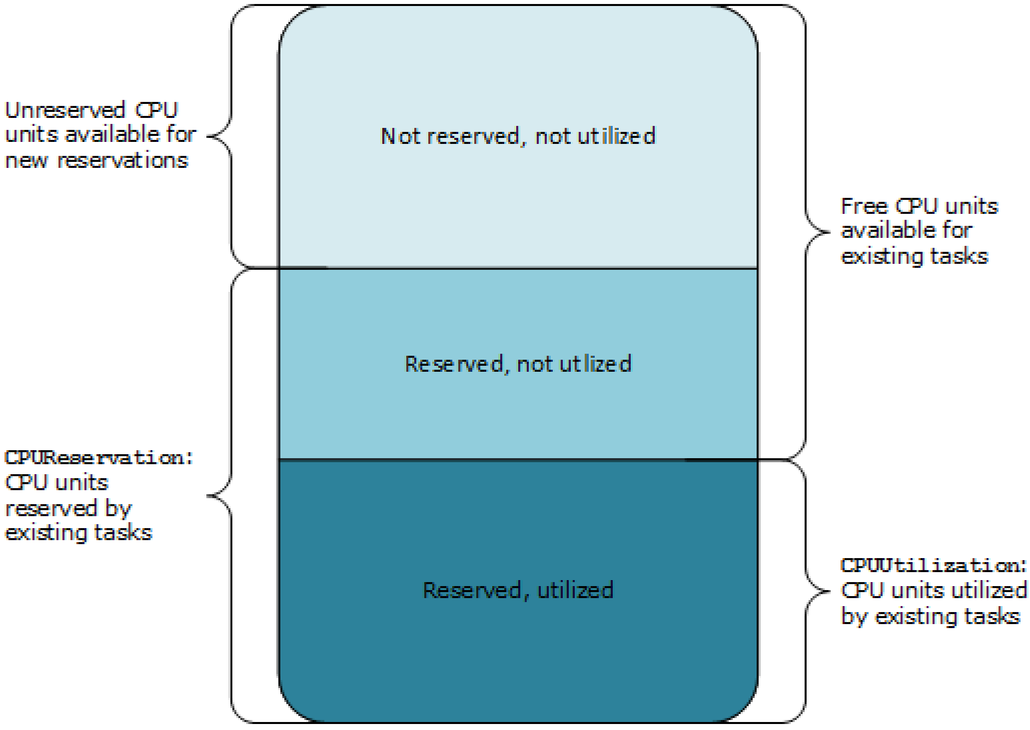
When you run a task in a cluster, Amazon ECS parses its task definition and reserves the aggregate CPU units, MiB of memory, and GPUs that are specified in its container definitions. Each minute, Amazon ECS calculates the number of CPU units, MiB of memory, and GPUs that are currently reserved for each task that is running in the cluster. The total amount of CPU, memory, and GPUs reserved for all tasks running on the cluster is calculated, and those numbers are reported to

CloudWatch as a percentage of the total registered resources for the cluster. If you specify a soft limit (`memoryReservation`) in the task definition, it's used to calculate the amount of reserved memory. Otherwise, the hard limit (`memory`) is used. The total MiB of memory reserved by tasks in a cluster also includes temporary file system (`tmpfs`) volume size and `sharedMemorySize` if defined in the task definition. For more information about hard and soft limits, shared memory size, and tmpfs volume size, see [Task Definition Parameters](#).

For example, a cluster has two active container instances registered: a `c4.4xlarge` instance and a `c4.large` instance. The `c4.4xlarge` instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The `c4.large` instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If a task definition reserves 1,024 CPU units and 2,048 MiB of memory, and ten tasks are started with this task definition on this cluster (and no other tasks are currently running), a total of 10,240 CPU units and 20,480 MiB of memory are reserved. This is reported to CloudWatch as 55% CPU reservation and 60% memory reservation for the cluster.

The following illustration shows the total registered CPU units in a cluster and what their reservation and utilization means to existing tasks and new task placement. The lower (Reserved, used) and center (Reserved, not used) blocks represent the total CPU units that are reserved for the existing tasks that are running on the cluster, or the `CPUReservation` CloudWatch metric. The lower block represents the reserved CPU units that the running tasks are actually using on the cluster, or the `CPUUtilization` CloudWatch metric. The upper block represents CPU units that are not reserved by existing tasks; these CPU units are available for new task placement. Existing tasks can use these unreserved CPU units as well, if their need for CPU resources increases. For more information, see the [`cpu`](#) task definition parameter documentation.



Aggregate cluster CPU units

Amazon ECS cluster utilization metrics

The cluster utilization metrics are available for CPU, memory, and, when there is an EBS volume attached to your tasks, EBS filesystem utilization. These metrics are only available for clusters with tasks or services hosted on Amazon EC2 instances. They're not supported on clusters with tasks hosted on AWS Fargate.

Amazon ECS cluster level CPU and memory utilization metrics

The CPU and memory utilization is measured as the percentage of CPU and memory that is used by all tasks on a cluster when compared to the aggregate CPU and memory that was registered for each active Amazon EC2 instances registered to the cluster. Only Amazon EC2 instances in `ACTIVE` or `DRAINING` status will affect cluster utilization metrics.

```
                                  (Total CPU units used by tasks in cluster) x 100
Cluster CPU utilization =
 --------------------------------------------------------------
                           (Total CPU units registered by container instances in
 cluster)
```

```
                                    (Total MiB of memory used by tasks in cluster x
 100)
Cluster memory utilization =
 ------------------------------------------------------------------
                             (Total MiB of memory registered by container instances in
 cluster)
```

Each minute, the Amazon ECS container agent on each Amazon EC2 instance calculates the number of CPU units and MiB of memory that is currently being used for each task that is running on that Amazon EC2 instance, and this information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster.

For example, a cluster has two active Amazon EC2 instances registered, a `c4.4xlarge` instance and a `c4.large` instance. The `c4.4xlarge` instance registers into the cluster with `16,384` CPU units and `30,158` MiB of memory. The `c4.large` instance registers with `2,048` CPU units and `3,768` MiB of memory. The aggregate resources of this cluster are `18,432` CPU units and `33,926` MiB of memory.

If ten tasks are running on this cluster and each task consumes `1,024` CPU units and `2,048` MiB of memory, a total of `10,240` CPU units and `20,480` MiB of memory are used on the cluster. This is reported to CloudWatch as 55% CPU utilization and 60% memory utilization for the cluster.

Amazon ECS cluster-level Amazon EBS filesystem utilization

The cluster level EBS filesystem utilization metric is measured as the total amount of the EBS filesystem in use by the tasks running on the cluster, divided by the total amount of EBS filesystem storage that was allocated for all of the tasks in the cluster.

```
                                        (Total GB of EBS filesystem used by tasks in
 cluster x 100)
Cluster EBS filesystem utilization =
 ---------------------------------------------------------------
                                        (Total GB of EBS filesystem allocated to tasks
 in cluster)
```

Amazon ECS service utilization metrics

The service utilization metrics are available for CPU, memory, and, when there is an EBS volume attached to your tasks, EBS filesystem utilization. The service level metrics are supported for services with tasks hosted on both Amazon EC2 instances and Fargate.

Service level CPU and memory utilization

The CPU and memory utilization is measured as the percentage of CPU and memory that is used by the Amazon ECS tasks that belong to a service on a cluster when compared to the CPU and memory that is specified in the service's task definition.

When viewing these metrics in CloudWatch, you can choose different statistics:

- **Average**: The average utilization across all tasks in the service. This is calculated using the formula below.
- **Minimum**: The utilization of the task with the lowest resource usage in the service. This represents the percentage of CPU or memory used by the least resource-intensive task compared to what was specified in the task definition.
- **Maximum**: The utilization of the task with the highest resource usage in the service. This represents the percentage of CPU or memory used by the most resource-intensive task compared to what was specified in the task definition.

The following formulas show how the Average statistic is calculated:

```
                                  (Total CPU units used by tasks in service) x 100
Service CPU utilization =
 --------------------------------------------------------------
                           (Total CPU units specified in the task definition) x (Number
 of tasks in the service)
```

```
                                     (Total MiB of memory used by tasks in service x
 100)
Service memory utilization =
 ------------------------------------------------------------------
                              (Total MiB of memory specified in the task definition)  x
 (Number of tasks in the service)
```

Note

The formulas above apply only to the Average statistic. The Minimum and Maximum statistics represent the individual task with the lowest and highest resource utilization, respectively, rather than an aggregate calculation across all tasks.

Amazon ECS collects metrics every 20 seconds. Each minute, the Amazon ECS container agent calculates the number of CPU units and MiB of memory that are currently being used for each running task owned by the service. This information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks owned by the service that are running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total resources that are specified for the service in the service's task definition. The minimum and maximum values are the smallest and largest of the 20 second metrics. The average is the aggregate of the 3 values.

If you specify a soft limit (`memoryReservation`), it's used to calculate the amount of reserved memory. Otherwise, the hard limit (`memory`) is used. For more information about hard and soft limits, see Task size.

For example, the task definition for a service specifies a total of 512 CPU units and 1,024 MiB of memory (with the hard limit `memory` parameter) for all of its containers. The service has a desired count of 1 running task, the service is running on a cluster with 1 `c4.large` container instance (with 2,048 CPU units and 3,768 MiB of total memory), and there are no other tasks running on the cluster. Although the task specifies 512 CPU units, because it is the only running task on a container instance with 2,048 CPU units, it can use up to four times the specified amount (2,048 / 512). However, the specified memory of 1,024 MiB is a hard limit and it can't be exceeded, so in this case, service memory utilization can't exceed 100%.

If the previous example used the soft limit `memoryReservation` instead of the hard limit `memory` parameter, the service's tasks could use more than the specified 1,024 MiB of memory as needed. In this case, the service's memory utilization could exceed 100%.

If your application has a sudden spike in memory utilization for a short amount of time, you will not see the service memory utilization increasing because Amazon ECS collects multiple data points every minute, and then aggregates them to one data point that is sent to CloudWatch.

If this task is performing CPU-intensive work during a period and using all 2,048 of the available CPU units and 512 MiB of memory, the service reports 400% CPU utilization and 50% memory utilization. If the task is idle and using 128 CPU units and 128 MiB of memory, the service reports 25% CPU utilization and 12.5% memory utilization.

> **Note**
>
> In this example, the CPU utilization will only go above 100% when the CPU units are defined at the container level. If you define CPU units at the task level, the utilization will not go above the defined task-level limit.

Service level EBS filesystem utilization

The service level EBS filesystem utilization is measured as the total amount of the EBS filesystem in use by the tasks that belong to the service, divided by the total amount of EBS filesystem storage that is allocated for all tasks that belong to the service.

Service `RUNNING` task count

You can use CloudWatch metrics to view the number of tasks in your services that are in the `RUNNING` state. For example, you can set a CloudWatch alarm for this metric to alert you if the number of running tasks in your service falls below a specified value.

Service `RUNNING` task count in Amazon ECS CloudWatch Container Insights

A "Number of Running Tasks" (`RunningTaskCount`) metric is available per cluster and per service when you use Amazon ECS CloudWatch Container Insights. You can use Container Insights for all new clusters created by opting in to the `containerInsights` account setting, on individual clusters by turning on the cluster settings during cluster creation, or on existing clusters by using the UpdateClusterSettings API. Metrics collected by CloudWatch Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see [CloudWatch Pricing](https://aws.amazon.com/cloudwatch/pricing/).

To view this metric, see [Amazon ECS Container Insights Metrics]() in the *Amazon CloudWatch User Guide*.

Amazon ECS service utilization metrics use cases

The following list provides information about when to use the Amazon ECS metrics.

- **Resource utilization monitoring**: Use average statistics to monitor CPU and memory consumption patterns, establish performance baselines, and detect gradual performance degradation trends.

- **Cost optimization**: Use average statistics to monitor resource usage, right-size containers, adjust reservations based on usage patterns, and implement scheduled scaling.
- **Performance benchmarking**: Use average statistics to compare metrics between service revisions, establish performance KPIs, and validate optimization improvements.
- **Resource floor detection**: Use average statistics to identify minimum resource needs during idle periods, set appropriate reservations, and detect abnormal drops.
- **Anomaly detection**: Use minimum statistics to spot unusual resource utilization drops that indicate potential problems, such as initialization failures or unexpected idle periods.
- **Scaling policy refinement**: Use minimum statistics to establish optimal scale-in thresholds based on minimum viable utilization to prevent aggressive scaling.
- **Capacity planning**: Use maximum statistics to set appropriate task sizes and plan infrastructure with sufficient headroom for traffic spikes.
- **Performance bottleneck identification**: Use maximum statistics to detect resource saturation points, identify bottlenecks, and determine when to increase task size.
- **Scaling policy configuration**: Use maximum statistics to set optimal scale-out thresholds based on peak patterns and configure burst capacity.
- **SLA compliance monitoring**: Use maximum statistics to track peak response times and error rates to ensure service performance meets defined SLAs.

Related metrics information

For information about Container Insights, see Container Insights use cases in the *CloudWatch User Guide*.

Automate responses to Amazon ECS errors using EventBridge

Using Amazon EventBridge, you can automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and what automated actions to take when an event matches a rule. The actions that can be automatically configured to include the following:

- Adding events to log groups in CloudWatch Logs
- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command

- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon Simple Queue Service (Amazon SQS) queue

For more information, see Getting started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

You can use Amazon ECS events for EventBridge to receive near real-time notifications regarding the current state of your Amazon ECS clusters. If your tasks are using the EC2 launch type, you can see the state of both the container instances and the current state of all tasks running on those container instances. If your tasks are using the Fargate launch type, you can see the state of the container instances.

Using EventBridge, you can build custom schedulers on top of Amazon ECS that are responsible for orchestrating tasks across clusters and monitoring the state of clusters in near real time. You can eliminate scheduling and monitoring code that continuously polls the Amazon ECS service for status changes and instead handle Amazon ECS state changes asynchronously using any EventBridge target. Targets might include AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, or Amazon Kinesis Data Streams.

An Amazon ECS event stream ensures that every event is delivered at least one time. If duplicate events are sent, the event provides enough information to identify duplicates. For more information, see Handling Amazon ECS events.

Events are relatively ordered, so that you can easily tell when an event occurred in relation to other events.

Topics

Amazon ECS events

Amazon ECS tracks the state of each of your tasks and services. If the state of a task or service changes, an event is generated and is sent to Amazon EventBridge. These events are classified as task state change events and service action events. These events and their possible causes are described in greater detail in the following sections.

Amazon ECS generates and sends the following types of events to EventBridge:

- Container instance state change
- Task state change
- Deployment state change
- Service action

> **Note**
>
> Amazon ECS might add other event types, sources, and details in the future. If you are deserializing event JSON data in code, make sure that your application is prepared to handle unknown properties to avoid issues if and when these additional properties are added.

In some cases, multiple events are generated for the same activity. For example, when a task is started on a container instance, a task state change event is generated for the new task. A container instance state change event is generated to account for the change in available resources, such as CPU, memory, and available ports, on the container instance. Likewise, if a container instance is terminated, events are generated for the container instance, the container agent connection status, and every task that was running on the container instance.

Container state change and task state change events contain two `version` fields: one in the main body of the event, and one in the `detail` object of the event. The following describes the differences between these two fields:

- The `version` field in the main body of the event is set to `0` on all events. For more information about EventBridge parameters, see [AWS service event metadata](#) in the *Amazon EventBridge User Guide*.
- The `version` field in the `detail` object of the event describes the version of the associated resource. Each time a resource changes state, this version is incremented. Because events can be sent multiple times, this field allows you to identify duplicate events. Duplicate events have the same version in the `detail` object. If you are replicating your Amazon ECS container instance and task state with EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the version reported in EventBridge for the resource (inside the `detail` object) to verify that the version in your event stream is current.

Service action events only contain the `version` field in the main body.

Service action events specify the service in 2 different fields:

- For events generated by `create-service`, the service is in the `serviceName` field.
- For events generated by `update-service`, the service is in the `service` field.

If you use automated tooling for service events, you need to code for both fields.

For information about how to create a rule for service action events, see Amazon ECS service action events.

For additional information about how to integrate Amazon ECS and EventBridge, see Integrating Amazon EventBridge and Amazon ECS.

Amazon ECS container instance state change events

The following scenarios cause container instance state change events:

You call the `StartTask`, `RunTask`, or `StopTask` API operations, either directly or with the AWS Management Console or SDKs.

> Placing or stopping tasks on a container instance modifies the available resources on the container instance, such as CPU, memory, and available ports.

The Amazon ECS service scheduler starts or stops a task.

> Placing or stopping tasks on a container instance modifies the available resources on the container instance, such as CPU, memory, and available ports.

The Amazon ECS container agent calls the `SubmitTaskStateChange` API operation with a `STOPPED` status for a task with a desired status of `RUNNING`.

> The Amazon ECS container agent monitors the state of tasks on your container instances, and it reports any state changes. If a task that is supposed to be `RUNNING` is transitioned to `STOPPED`, the agent releases the resources that were allocated to the stopped task, such as CPU, memory, and available ports.

You deregister the container instance with the `DeregisterContainerInstance` API operation, either directly or with the AWS Management Console or SDKs.

> Deregistering a container instance changes the status of the container instance and the connection status of the Amazon ECS container agent.

A task was stopped when an EC2 instance was stopped.

When you stop a container instance, the tasks that are running on it are transitioned to the `STOPPED` status.

The Amazon ECS container agent registers a container instance for the first time.

The first time the Amazon ECS container agent registers a container instance (at launch or when first run manually), this creates a state change event for the instance.

The Amazon ECS container agent connects or disconnects from Amazon ECS.

When the Amazon ECS container agent connects or disconnects from the Amazon ECS backend, it changes the `agentConnected` status of the container instance.

> **Note**
>
> The Amazon ECS container agent disconnects and reconnects several times per hour as a part of its normal operation, so agent connection events should be expected. These events are not an indication that there is an issue with the container agent or your container instance.

You upgrade the Amazon ECS container agent on an instance.

The container instance detail contains an object for the container agent version. If you upgrade the agent, this version information changes and generates an event.

Example Container instance state change event

Container instance state change events are delivered in the following format. The `detail` section below resembles the [ContainerInstance](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_ContainerInstance.html) object that is returned from a [DescribeContainerInstances](https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_DescribeContainerInstances.html) API operation in the *Amazon Elastic Container Service API Reference*. For more information about EventBridge parameters, see [AWS service event metadata](https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-events.html) in the *Amazon EventBridge User Guide*.

```
{
  "version": "0",
  "id": "8952ba83-7be2-4ab5-9c32-6687532d15a2",
  "detail-type": "ECS Container Instance State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2016-12-06T16:41:06Z",
```

```
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315"
  ],
  "detail": {
    "agentConnected": true,
    "attributes": [
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
      },
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role-network-host"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
      },
      {
        "name": "com.amazonaws.ecs.capability.privileged-container"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
      },
      {
        "name": "com.amazonaws.ecs.capability.ecr-auth"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.20"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.21"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.22"
      },
```

```
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.23"
      },
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role"
      }
    ],
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315",
    "ec2InstanceId": "i-f3a8506b",
    "registeredResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 2048
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 3767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
          "22",
          "2376",
          "2375",
          "51678",
          "51679"
        ]
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "remainingResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 1988
```

```
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
          "22",
          "2376",
          "2375",
          "51678",
          "51679"
        ]
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "status": "ACTIVE",
    "version": 14801,
    "versionInfo": {
      "agentHash": "aebcbca",
      "agentVersion": "1.13.0",
      "dockerVersion": "DockerVersion: 1.11.2"
    },
    "updatedAt": "2016-12-06T16:41:06.991Z"
  }
}
```

Amazon ECS task state change events

The following scenarios cause task state change events:

You call the `StartTask`, `RunTask`, or `StopTask` API operations, either directly or with the AWS Management Console, AWS CLI, or SDKs.

> Starting or stopping tasks creates new task resources or modifies the state of existing task resources.

The Amazon ECS service scheduler starts or stops a task.

Starting or stopping tasks creates new task resources or modifies the state of existing task resources.

The Amazon ECS container agent calls the `SubmitTaskStateChange` API operation.

For the Amazon EC2 launch type, the Amazon ECS container agent monitors the state of your tasks on your container instances. The Amazon ECS container agent reports any state changes. State changes might include changes from `PENDING` to `RUNNING` or from `RUNNING` to `STOPPED`.

You force deregistration of the underlying container instance with the `DeregisterContainerInstance` API operation and the `force` flag, either directly or with the AWS Management Console or SDKs.

Deregistering a container instance changes the status of the container instance and the connection status of the Amazon ECS container agent. If tasks are running on the container instance, the `force` flag must be set to allow deregistration. This stops all tasks on the instance.

The underlying container instance is stopped or terminated.

When you stop or terminate a container instance, the tasks that are running on it are transitioned to the `STOPPED` status.

A container in the task changes state.

The Amazon ECS container agent monitors the state of containers within tasks. For example, if a container that is running within a task stops, this container state change generates an event.

A task using the Fargate Spot capacity provider receives a termination notice.

When a task is using the `FARGATE_SPOT` capacity provider and is stopped due to a Spot interruption, a task state change event is generated.

Example Task state change event

Task state change events are delivered in the following format. Note the following about the fields:

- The health status of the event is not available in the task state change event. If you need the task health status, you can run [describe-tasks](#).

- When your containers use an image hosted with Amazon ECR, the `imageDigest` field is returned.
- The values for the `createdAt`, `connectivityAt`, `pullStartedAt`, `startedAt`, `pullStoppedAt`, and `updatedAt` fields are ISO string timestamps.
- The `detail-type` value is "ECS Task State Change".
- When the event is generated for a stopped task, the `stoppedReason` and `stopCode` fields provide additional information about why the task stopped (for example, "User initiated").

For more information about EventBridge parameters, see AWS service event metadata in the *Amazon EventBridge Events Reference*.

For information about how to configure an Amazon EventBridge event rule that only captures task events where the task has stopped running because one of its essential containers has terminated, see Sending Amazon Simple Notification Service alerts for Amazon ECS task stopped events


```
{
    "version": "0",
    "id": "105f6bb1-4da6-c630-4965-35383018cbca",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2025-05-06T11:02:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/a1173316d40a45dea9"
    ],
    "detail": {
        "attachments": [
            {
                "id": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-0d0eab1bb38d5ca64"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-0103a2f01bad57d71"
```

```
                    },
                    {
                        "name": "macAddress",
                        "value": "0e:50:d1:c1:77:81"
                    },
                    {
                        "name": "privateDnsName",
                        "value": "ip-10-0-1-163.ec2.internal"
                    },
                    {
                        "name": "privateIPv4Address",
                        "value": "10.0.1.163"
                    }
                ]
            }
        ],
        "attributes": [
            {
                "name": "ecs.cpu-architecture",
                "value": "x86_64"
            }
        ],
        "availabilityZone": "us-east-1b",
        "capacityProviderName": "FARGATE",
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/example-cluster",
        "connectivity": "CONNECTED",
        "connectivityAt": "2025-05-06T11:02:17.19Z",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/a0a99b87-baa8-4bf6-b9f1-a9a95917a635",
                "lastStatus": "RUNNING",
                "name": "web",
                "image": "nginx",
                "imageDigest":
 "sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66",
                "runtimeId": "a1173316d40a45dea9-0265927825",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
```

```
                ],
                "cpu": "99",
                "memory": "100"
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/a2010e2d-ba7c-4135-8b79-e0290ff3cd8c",
                "lastStatus": "RUNNING",
                "name": "aws-guardduty-agent-nm40lC",
                "imageDigest":
 "sha256:bf9197abdf853607e5fa392b4f97ccdd6ca56dd179be3ce8849e552d96582ac8",
                "runtimeId": "a1173316d40a45dea9-2098416933",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "null"
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example-
cluster/a1173316d40a45dea9/dccf0ca2-d929-471f-a5c3-98006fd4379e",
                "lastStatus": "RUNNING",
                "name": "aws-otel-collector",
                "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.32.0",
                "imageDigest":
 "sha256:7a1b3560655071bcacd66902c20ebe9a69470d5691fe3bd36baace7c2f3c4640",
                "runtimeId": "a1173316d40a45dea9-4027662657",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
                "networkInterfaces": [
                    {
                        "attachmentId": "fe3a9a46-6a47-40ee-afd9-7952ae90a75a",
                        "privateIpv4Address": "10.0.1.163"
                    }
                ],
                "cpu": "0"
            }
        ],
        "cpu": "256",
        "createdAt": "2025-05-06T11:02:13.877Z",
```

```
        "desiredStatus": "RUNNING",
        "enableExecuteCommand": false,
        "ephemeralStorage": {
            "sizeInGiB": 20
        },
        "group": "family:webserver",
        "launchType": "FARGATE",
        "lastStatus": "RUNNING",
        "memory": "512",
        "overrides": {
            "containerOverrides": [
                {
                    "name": "web"
                },
                {
                    "environment": [
                        {
                            "name": "CLUSTER_NAME",
                            "value": "example-cluster"
                        },
                        {
                            "name": "REGION",
                            "value": "us-east-1"
                        },
                        {
                            "name": "HOST_PROC",
                            "value": "/host_proc"
                        },
                        {
                            "name": "AGENT_RUNTIME_ENVIRONMENT",
                            "value": "ecsfargate"
                        },
                        {
                            "name": "STAGE",
                            "value": "prod"
                        }
                    ],
                    "memory": 128,
                    "name": "aws-guardduty-agent-nm40lC"
                },
                {
                    "name": "aws-otel-collector"
                }
            ]
```

```
        },
        "platformVersion": "1.4.0",
        "pullStartedAt": "2025-05-06T11:02:24.162Z",
        "pullStoppedAt": "2025-05-06T11:02:33.493Z",
        "startedAt": "2025-05-06T11:02:34.325Z",
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example-cluster/
a1173316d40a45dea9",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/
webserver:5",
        "updatedAt": "2025-05-06T11:02:34.325Z",
        "version": 3
    }
}
```

Example

The following is an example of a task state change event for the EC2 launch type.

```
{
    "version": "0",
    "id": "a65cf262-f104-0dd5-ceda-4b09ba71a441",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "123456789012",
    "time": "2025-05-12T13:12:06Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ecs:us-east-1:123456789012:task/example/c1ffa94f19a540ed8d9f7e1d2a5d"
    ],
    "detail": {
        "attachments": [
            {
                "id": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-0d0eab1bb38d5ca64"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-0ea90f746500773a4"
                    },
```

```
                    {
                        "name": "macAddress",
                        "value": "0e:d5:9b:ce:49:fb"
                    },
                    {
                        "name": "privateDnsName",
                        "value": "ip-10-0-1-37.ec2.internal"
                    },
                    {
                        "name": "privateIPv4Address",
                        "value": "10.0.1.37"
                    }
                ]
            }
        ],
        "attributes": [
            {
                "name": "ecs.cpu-architecture",
                "value": "x86_64"
            }
        ],
        "availabilityZone": "us-east-1b",
        "capacityProviderName": "Infra-ECS-Cluster-example-fa84e0cc-
AsgCapacityProvider-OseQJU9pizmp",
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/example",
        "connectivity": "CONNECTED",
        "connectivityAt": "2025-05-12T13:11:44.98Z",
        "containerInstanceArn": "arn:aws:ecs:us-east-1:123456789012:container-instance/
example/d1d84798400f49f3b21cb61610c1e",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example/
c1ffa94f19a540ed8d9f7e1d2a5d3626/197d0994-5367-4a6d-9f9a-f075e4a6",
                "lastStatus": "RUNNING",
                "name": "aws-otel-collector",
                "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.32.0",
                "imageDigest":
 "sha256:7a1b3560655071bcacd66902c20ebe9a69470d5691fe3bd36baace7c2f3c4640",
                "runtimeId":
 "8e926f0ccd8fe2b459926f49584ba6d33a3d9f61398dbabe944ee6a13a8ff3a1",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
                "networkInterfaces": [
                    {
```

```
                        "attachmentId": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                        "privateIpv4Address": "10.0.1.37"
                    }
                ],
                "cpu": "0"
            },
            {
                "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/example/
c1ffa94f19a540ed8d9f7e1d2a5d3626/cab39ef0-9c50-459d-844b-b9d51d73d",
                "lastStatus": "RUNNING",
                "name": "web",
                "image": "nginx",
                "imageDigest":
 "sha256:c15da6c91de8d2f436196f3a768483ad32c258ed4e1beb3d367a27ed67253e66",
                "runtimeId":
 "9f1c73f0094f051541d9e5c2ab1e172d83c4eb5171bcc857c4504b02770ff3b8",
                "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
                "networkInterfaces": [
                    {
                        "attachmentId": "52333e3b-b812-41a8-b057-9ed184bbe5e1",
                        "privateIpv4Address": "10.0.1.37"
                    }
                ],
                "cpu": "99",
                "memory": "100"
            }
        ],
        "cpu": "256",
        "createdAt": "2025-05-12T13:11:44.98Z",
        "desiredStatus": "RUNNING",
        "enableExecuteCommand": false,
        "group": "family:webserver",
        "launchType": "EC2",
        "lastStatus": "RUNNING",
        "memory": "512",
        "overrides": {
            "containerOverrides": [
                {
                    "name": "aws-otel-collector"
                },
                {
                    "name": "web"
                }
```

```
            ]
        },
        "pullStartedAt": "2025-05-12T13:11:59.491Z",
        "pullStoppedAt": "2025-05-12T13:12:05.896Z",
        "startedAt": "2025-05-12T13:12:06.053Z",
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/example/
c1ffa94f19a540ed8d9f7e1d2a5d",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/
webserver",
        "updatedAt": "2025-05-12T13:12:06.053Z",
        "version": 4
    }
}
```

Amazon ECS service action events

Amazon ECS sends service action events with the detail type **ECS Service Action**. Unlike the container instance and task state change events, the service action events do not include a version number in the `details` response field. The following is an event pattern that is used to create an EventBridge rule for Amazon ECS service action events. For more information, see [Getting started with EventBridge](#) in the *Amazon EventBridge User Guide*.

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Service Action"
    ]
}
```

Amazon ECS sends events with `INFO`, `WARN`, and `ERROR` event types. The following are the service action events.

Service action events with INFO event type

`SERVICE_STEADY_STATE`

The service is healthy and at the desired number of tasks, thus reaching a steady state. The service scheduler reports the status periodically, so you might receive this message multiple times.

`TASKSET_STEADY_STATE`

The task set is healthy and at the desired number of tasks, thus reaching a steady state.

`CAPACITY_PROVIDER_STEADY_STATE`

A capacity provider associated with a service reaches a steady state.

`SERVICE_DESIRED_COUNT_UPDATED`

When the service scheduler updates the computed desired count for a service or task set. This event is not sent when the desired count is manually updated by a user.

`TASKS_STOPPED`

The service has stopped the running task.

`SERVICE_DEPLOYMENT_IN_PROGRESS`

A service deployment is in progress. The service deployment can be either a rollback, or a new service revision.

`SERVICE_DEPLOYMENT_COMPLETED`

A service deployment is in the steady state and is complete. The service deployment can be either a rollback, or to deploy an updated service revision.

Service action events with `WARN` event type

`SERVICE_TASK_START_IMPAIRED`

The service is unable to consistently start tasks successfully.

`SERVICE_DISCOVERY_INSTANCE_UNHEALTHY`

A service using service discovery contains an unhealthy task. The service scheduler detects that a task within a service registry is unhealthy.

`VPC_LATTICE_TARGET_UNHEALTHY`

The service using VPC Lattice has detected one of the targets for the VPC Lattice is unhealthy.

Service action events with ERROR event type

`SERVICE_DAEMON_PLACEMENT_CONSTRAINT_VIOLATED`

A task in a service using the `DAEMON` service scheduler strategy no longer meets the placement constraint strategy for the service.

`ECS_OPERATION_THROTTLED`

The service scheduler has been throttled due to the Amazon ECS API throttle limits.

`SERVICE_DISCOVERY_OPERATION_THROTTLED`

The service scheduler has been throttled due to the AWS Cloud Map API throttle limits. This can occur on services configured to use service discovery.

`SERVICE_TASK_PLACEMENT_FAILURE`

The service scheduler is unable to place a task. The cause will be described in the `reason` field.

A common cause for this service event being generated is because of a lack of resources in the cluster to place the task. For example, not enough CPU or memory capacity on the available container instances or no container instances being available. Another common cause is when the Amazon ECS container agent is disconnected on the container instance, causing the scheduler to be unable to place the task.

`SERVICE_TASK_CONFIGURATION_FAILURE`

The service scheduler is unable to place a task due to a configuration error. The cause will be described in the `reason` field.

A common cause of this service event being generated is because tags were being applied to the service but the user or role had not opted in to the new Amazon Resource Name (ARN) format in the Region. For more information, see Amazon Resource Names (ARNs) and IDs. Another common cause is that Amazon ECS was unable to assume the task IAM role provided.

`SERVICE_HEALTH_UNKNOWN`

The service was unable to describe the health data for tasks.

`SERVICE_DEPLOYMENT_FAILED`

A service deployment did not reach the steady. This happens when a CloudWatch is triggered or the circuit breaker detects a service deployment failure.

Example Service steady state event

Service steady state events are delivered in the following format. For more information about EventBridge parameters, see [Events in EventBridge](#) in the *Amazon EventBridge User Guide*.

```
{
    "version": "0",
    "id": "af3c496d-f4a8-65d1-70f4-a69d52e9b584",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:27:22Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:27:22.695Z"
    }
}
```

Example Capacity provider steady state event

Capacity provider steady state events are delivered in the following format.

```
{
    "version": "0",
    "id": "b9baa007-2f33-0eb1-5760-0d02a572d81f",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:37:00Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "CAPACITY_PROVIDER_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
```

```
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/ASG-tutorial-
capacity-provider"
        ],
        "createdAt": "2019-11-19T19:37:00.807Z"
    }
}
```

Example Service task start impaired event

Service task start impaired events are delivered in the following format.

```
{
    "version": "0",
    "id": "57c9506e-9d21-294c-d2fe-e8738da7e67d",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "WARN",
        "eventName": "SERVICE_TASK_START_IMPAIRED",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:55:38.725Z"
    }
}
```

Example Service task placement failure event

Service task placement failure events are delivered in the following format. For more information, see [Events in EventBridge](#) in the *Amazon EventBridge User Guide*.

In the following example, the task was attempting to use the `FARGATE_SPOT` capacity provider but the service scheduler was unable to acquire any Fargate Spot capacity.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
```

```
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "ERROR",
        "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/FARGATE_SPOT"
        ],
        "reason": "RESOURCE:FARGATE",
        "createdAt": "2019-11-06T19:09:33.087Z"
    }
}
```

In the following example for the EC2 launch type, the task was attempted to launch on the Container Instance `2dd1b186f39845a584488d2ef155c131` but the service scheduler was unable to place the task because of insufficient CPU.

```
{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
  "detail-type": "ECS Service Action",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2019-11-19T19:55:38Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "ERROR",
    "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
    "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
    "containerInstanceArns": [
    "arn:aws:ecs:us-west-2:111122223333:container-instance/
default/2dd1b186f39845a584488d2ef155c131"
    ],
```

```
    "reason": "RESOURCE:CPU",
    "createdAt": "2019-11-06T19:09:33.087Z"
  }
}
```

Amazon ECS service deployment state change events

Amazon ECS sends service deployment change state events with the detail type **ECS Deployment State Change**. The following is an event pattern that is used to create an EventBridge rule for Amazon ECS service deployment state change events. For more information about creating an EventBridge rule, see [Getting started with Amazon EventBridge](#) in the *Amazon EventBridge User Guide*.

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Deployment State Change"
    ]
}
```

Amazon ECS sends events with `INFO` and `ERROR` event types. For more information, see [Amazon ECS service action events](#)

The following are the service deployment state change events.

`SERVICE_DEPLOYMENT_IN_PROGRESS`

The service deployment is in progress. This event is sent for both initial deployments and rollback deployments.

`SERVICE_DEPLOYMENT_COMPLETED`

The service deployment has completed. This event is sent once a service reaches a steady state after a deployment.

`SERVICE_DEPLOYMENT_FAILED`

The service deployment has failed. This event is sent for services with deployment circuit breaker logic turned on.

Example service deployment in progress event

Service deployment in progress events are delivered when both an initial and a rollback deployment is started. The difference between the two is in the `reason` field. For more information about EventBridge parameters, see [AWS service event metadata;](#) in the *Amazon EventBridge User Guide*.

The following shows an example output for an initial deployment starting.

```
{
   "version": "0",
   "id": "ddca6449-b258-46c0-8653-e0e3a6EXAMPLE",
   "detail-type": "ECS Deployment State Change",
   "source": "aws.ecs",
   "account": "111122223333",
   "time": "2020-05-23T12:31:14Z",
   "region": "us-west-2",
   "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
   ],
   "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment deploymentId in progress."
   }
}
```

The following shows an example output for a rollback deployment starting. The `reason` field provides the ID of the deployment the service is rolling back to.

```
{
   "version": "0",
   "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
   "detail-type": "ECS Deployment State Change",
   "source": "aws.ecs",
   "account": "111122223333",
   "time": "2020-05-23T12:31:14Z",
   "region": "us-west-2",
   "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
```

```
    ],
    "detail": {
         "eventType": "INFO",
         "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
         "deploymentId": "ecs-svc/123",
         "updatedAt": "2020-05-23T11:11:11Z",
         "reason": "ECS deployment circuit breaker: rolling back to
 deploymentId deploymentID."
    }
}
```

Example service deployment completed event

Service deployment completed state events are delivered in the following format. For more information, see Deploy Amazon ECS services by replacing tasks.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
         "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
         "eventType": "INFO",
         "eventName": "SERVICE_DEPLOYMENT_COMPLETED",
         "deploymentId": "ecs-svc/123",
         "updatedAt": "2020-05-23T11:11:11Z",
         "reason": "ECS deployment deploymentID completed."
    }
}
```

Example service deployment failed event

Service deployment failed state events are delivered in the following format. A service deployment failed state event will only be sent for services that have deployment circuit breaker logic turned on. For more information, see Deploy Amazon ECS services by replacing tasks.

```
{
```

```
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
         "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
         "eventType": "ERROR",
         "eventName": "SERVICE_DEPLOYMENT_FAILED",
         "deploymentId": "ecs-svc/123",
         "updatedAt": "2020-05-23T11:11:11Z",
         "reason": "ECS deployment circuit breaker: task failed to start."
    }
}
```

Handling Amazon ECS events

Amazon ECS sends events on an *at least once* basis. This means you might receive multiple copies of a given event. Additionally, events may not be delivered to your event listeners in the order in which the events occurred.

To order of events properly, the `detail` section of each event contains a `version` property. Each time a resource changes state, this `version` is incremented. Duplicate events have the same `version` in the `detail` object. If you are replicating your Amazon ECS container instance and task state with EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the `version` reported in EventBridge for the resource to verify that the version in your event stream is current. Events with a higher version property number should be treated as occurring later than events with lower version numbers.

Example: Handling events in an AWS Lambda function

The following example shows a Lambda function written in Python 3.9 that captures both task and container instance state change events and saves them to one of two Amazon DynamoDB tables:

- *ECSCtrInstanceState* – Stores the latest state for a container instance. The table ID is the `containerInstanceArn` value of the container instance.
- *ECSTaskState* – Stores the latest state for a task. The table ID is the `taskArn` value of the task.

```
import json
import boto3

def lambda_handler(event, context):
    id_name = ""
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print((json.dumps(event)))

    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type
 of: aws.ecs")

    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
        id_name = "taskArn"
        event_id = event["detail"]["taskArn"]
    elif event["detail-type"] == "ECS Container Instance State Change":
        table_name = "ECSCtrInstanceState"
        id_name =  "containerInstanceArn"
        event_id = event["detail"]["containerInstanceArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting
 without saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")


    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
```

```
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print(("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling"))
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
 updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event -
 ignoring")
    else:
        print(("Saving new event - ID " + event_id))

        table.put_item(
            Item=new_record
        )
```

The following Fargate example shows a Lambda function written in Python 3.9 that captures task state change events and saves them to the following Amazon DynamoDB table:

```
import json
import boto3

def lambda_handler(event, context):
    id_name = ""
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print((json.dumps(event)))

    if event["source"] != "aws.ecs":
       raise ValueError("Function only supports input from events with a source type
 of: aws.ecs")
```

```
    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
        id_name = "taskArn"
        event_id = event["detail"]["taskArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting
 without saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")


    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print(("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling"))
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
 updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event -
 ignoring")
```

```
    else:
        print(("Saving new event - ID " + event_id))

        table.put_item(
            Item=new_record
        )
```

Monitor Amazon ECS containers using Container Insights with enhanced observability

Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. Container Insights with enhanced observability provides all the Container Insights metrics, plus additional task and container metrics.

Container Insights discovers all the running containers in a cluster and collects performance data at every layer of the performance stack. Operational data is collected as performance log events. These are entries that use a structured JSON schema for high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates higher-level aggregated metrics at the cluster, service, and task level as CloudWatch metrics. The metrics include utilization for resources such as CPU, memory, disk, and network. The metrics are available in CloudWatch automatic dashboards.

The metrics only reflect the resources with running tasks during the specified time range. For example, if you have a cluster with one service in it but that service has no tasks in a `RUNNING` state, there will be no metrics sent to CloudWatch. If you have two services and one of them has running tasks and the other doesn't, only the metrics for the service with running tasks will be sent.

On December 2, 2024, AWS released Container Insights with enhanced observability for Amazon ECS. This version supports enhanced observability for Amazon ECS clusters using the Amazon EC2 and Fargate launch types. After you configure Container Insights with enhanced observability on Amazon ECS, Container Insights auto-collects detailed infrastructure telemetry from the cluster level down to the container level in your environment and displays these critical performance data in curated dashboards removing the heavy lifting in observability set-up. For information about how to set up Container Insights with enhanced observability, see [Container Insights with enhanced observability](#).

We recommend that you use Container Insights with enhanced observability instead of Container Insights because it provides detailed visibility in your container environment, reducing the mean time to resolution. For more information, see [Amazon ECS Container Insights with enhanced observability metrics](#) in the *Amazon CloudWatch User Guide*.

The events that you can view are the ones that Amazon ECS sends to Amazon EventBridge. For more information, see Amazon ECS events.

> **Important**
>
> Metrics collected by CloudWatch Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see CloudWatch Pricing.

Determine Amazon ECS task health using container health checks

When you create a task definition, you can configure a health check for your containers. Health checks are commands that run locally on a container and validate application health and availability.

The Amazon ECS container agent only monitors and reports on the health checks that are specified in the task definition. Amazon ECS doesn't monitor Docker health checks that are embedded in a container image but aren't specified in the container definition. Health check parameters that are specified in a container definition override any Docker health checks that exist in the container image.

When a health check is defined in a task definition, the container runs the health check process inside the container, and then evaluate the exit code to determine the application health.

The health check consists the following parameters:

- Command – The command that the container runs to determine if it's healthy. The string array can start with `CMD` to run the command arguments directly, or `CMD-SHELL` to run the command with the container's default shell. Use `CMD-SHELL` when you need shell features like pipes, redirects, command chaining, or environment variable expansion. For example, `CMD-SHELL` allows you to use commands like `curl -f http://localhost/ || exit 1` where the shell interprets the `||` operator. Use `CMD` for simple commands that don't require shell interpretation.
- Interval – The period of time (in seconds) between each health check.
- Timeout – The period of time (in seconds) to wait for a health check to succeed before it's considered a failure.
- Retries – The number of times to retry a failed health check before the container is considered unhealthy.

- Start period – The optional grace period to provide containers time to bootstrap in before failed health checks count towards the maximum number of retries.

  If a health check succeeds within the `startPeriod`, then the container is considered healthy and any subsequent failures count toward the maximum number of retries.

For information about how to specify a health check in a task definition, see Health check.

The following describes the possible health status values for a container:

- `HEALTHY`–The container health check has passed successfully.
- `UNHEALTHY`–The container health check has failed.
- `UNKNOWN`–The container health check is being evaluated, there's no container health check defined, or Amazon ECS doesn't have the health status of the container.

The health check commands run on the container. Therefore you must include the commands in the container image.

The health check connects to the application through the container's loopback interface at `localhost` or `127.0.0.1`. An exit code of `0` indicates success, and non-zero exit code indicates failure.

Consider the following when using container health checks:

- Container health checks require version 1.17.0 or greater of the Amazon ECS container agent.
- Container health checks are supported for Fargate tasks if you're using Linux platform version `1.1.0` or greater or Windows platform version `1.1.0` or greater

How Amazon ECS determines task health

Containers that are essential and have health check command in the task definition are the only ones considered to determine the task health.

The following rules are evaluated in order:

1. If the status of one essential container is `UNHEALTHY`, then the task status is `UNHEALTHY`.
2. If the status of one essential container is `UNKNOWN`, then the task status is `UNKNOWN`.

3. If the status of all essential containers are HEALTHY, then the task status is HEALTHY.

Consider the following task health example with 2 essential containers.

Container 1 health	Container 2 health	Task health
UNHEALTHY	UNKNOWN	UNHEALTHY
UNHEALTHY	HEALTHY	UNHEALTHY
HEALTHY	UNKNOWN	UNKNOWN
HEALTHY	HEALTHY	HEALTHY

Consider the following task health example with 3 containers.

Container 1 health	Container 2 health	Container 3 health	Task health
UNHEALTHY	UNKNOWN	UNKNOWN	UNHEALTHY
UNHEALTHY	UNKNOWN	HEALTHY	UNHEALTHY
UNHEALTHY	HEALTHY	HEALTHY	UNHEALTHY
HEALTHY	UNKNOWN	HEALTHY	UNKNOWN
HEALTHY	UNKNOWN	UNKNOWN	UNKNOWN
HEALTHY	HEALTHY	HEALTHY	HEALTHY

How health checks are affected by agent disconnects

If the Amazon ECS container agent becomes disconnected from the Amazon ECS service, this won't cause a container to transition to an UNHEALTHY status. This is by design, to ensure that containers remain running during agent restarts or temporary unavailability. The health check status is the "last heard from" response from the Amazon ECS agent, so if the container was considered HEALTHY prior to the disconnect, that status will remain until the agent reconnects and another health check occurs. There are no assumptions made about the status of the container health checks.

Viewing Amazon ECS container health

You can view the container health in the console, and using the API in the `DescribeTasks` response. For more information, see DescribeTasks in the *Amazon Elastic Container Service API Reference*.

If you use logging for your container, for example Amazon CloudWatch Logs, you can configure the health check command to forward the container health output to your logs. Make sure to use `2&1` to catch both the `stdout` and `stderr` information. The following example uses `CMD-SHELL` because it requires shell features like the pipe (>>) and logical OR (||) operators:

```
"command": [
     "CMD-SHELL",
     "curl -f http://localhost/ >> /proc/1/fd/1 2>&1  || exit 1"
   ],
```

Monitor Amazon ECS container instance health

Amazon ECS provides container instance health monitoring. You can quickly determine whether Amazon ECS has detected any problems that might prevent your container instances from running containers. Amazon ECS performs automated checks on every running container instance with agent version `1.57.0` or later to identify issues. For more information on verifying the agent version an a container instance, see Updating the Amazon ECS container agent.

You must be using AWS CLI version `1.22.3` or later or AWS CLI version `2.3.6` or later. For information about how to update the AWS CLI, see Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide Version 2*.

To view the container instance health, run `describe-container-instances` with the `CONTAINER_INSTANCE_HEALTH` option.

The following are the valid values for `overallStatus` :

- `OK`
- `IMPAIRED`
- `INSUFFICIENT_DATA`

- INITIALIZING

The following is an example of how to run `describe-container-instances`.

```
aws ecs describe-container-instances \
     --cluster cluster_name \
     --container-instances 47279cd2cadb41cbaef2dcEXAMPLE \
     --include CONTAINER_INSTANCE_HEALTH
```

The following is an example of the health status object in the output.

```
"healthStatus": {
 "overallStatus": "OK",
 "details": [{
  "type": "CONTAINER_RUNTIME",
  "status": "OK",
  "lastUpdated": "2021-11-10T03:30:26+00:00",
  "lastStatusChange": "2021-11-10T03:26:41+00:00"
 }]
}
```

Container instance-health issues

When the `overallStatus` any status other than `OK`, try the following:

- Wait, and then run `describe-container-instances`
- View your container instance health in the EC2 console or by using the CLI.
- Review the CloudWatch metrics. For more information, see Monitor Amazon ECS using CloudWatch
- Check the AWS Health Dashboard to see if there are any issues with the service.

Identify Amazon ECS optimization opportunities using application trace data

Amazon ECS integrates with AWS Distro for OpenTelemetry to collect trace data from your application. Amazon ECS uses an AWS Distro for OpenTelemetry sidecar container to collect and route trace data to AWS X-Ray. For more information, see Setting up AWS Distro for

OpenTelemetry Collector in Amazon ECS. You can then use AWS X-Ray to identify errors and exceptions, analyze performance bottlenecks and response times.

For the AWS Distro for OpenTelemetry Collector to send trace data to AWS X-Ray, your application must be configured to create the trace data. For more information, see Instrumenting your application for AWS X-Ray in the *AWS X-Ray Developer Guide*.

Required IAM permissions for AWS Distro for OpenTelemetry integration with AWS X-Ray

The Amazon ECS integration with AWS Distro for OpenTelemetry requires that you create a task role and specify the role in your task definition. We recommend that you configure the AWS Distro for OpenTelemetry sidecar to route container logs to CloudWatch Logs.

> **Important**
>
> If you also collect application metrics using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see Correlate Amazon ECS application performance using application metrics.

Create the following policy, and then attach it to the task execution role.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:PutRetentionPolicy",
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.

8. Choose **Create policy** to save your new policy.

Specifying the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration in your task definition

The Amazon ECS console simplifies creating the AWS Distro for OpenTelemetry sidecar container by using the **Use trace collection** option. For more information, see Creating an Amazon ECS task definition using the console.

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition. The following task definition snippet shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration.

```
{
 "family": "otel-using-xray",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryXrayRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [{
   "name": "aws-otel-emitter",
   "image": "application-image",
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "true",
     "awslogs-group": "/ecs/aws-otel-emitter",
     "awslogs-region": "us-east-1",
     "awslogs-stream-prefix": "ecs"
    }
   },
   "dependsOn": [{
    "containerName": "aws-otel-collector",
    "condition": "START"
   }]
  },
  {
   "name": "aws-otel-collector",
   "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
   "essential": true,
   "command": [
    "--config=/etc/ecs/otel-instance-metrics-config.yaml"
   ],
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
```

```
      "awslogs-create-group": "True",
      "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
     }
    }
   }
  ],
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
   "FARGATE"
  ],
  "cpu": "1024",
  "memory": "3072"
}
```

Correlate Amazon ECS application performance using application metrics

Amazon ECS on Fargate supports collecting metrics from your applications running on Fargate and exporting them to either Amazon CloudWatch or Amazon Managed Service for Prometheus.

You can use the collected metadata to correlate application performance data with underlying infrastructure data, reducing the mean time to resolve the problem.

Amazon ECS uses an AWS Distro for OpenTelemetry sidecar container to collect and route your application metrics to the destination. The Amazon ECS console experience simplifies the process of adding this integration when creating your task definitions.

Topics

- Exporting application metrics to Amazon CloudWatch
- Exporting application metrics to Amazon Managed Service for Prometheus

Exporting application metrics to Amazon CloudWatch

Amazon ECS on Fargate supports exporting your custom application metrics to Amazon CloudWatch as custom metrics. This is done by adding the AWS Distro for OpenTelemetry sidecar container to your task definition. The Amazon ECS console simplifies this process by adding the **Use**

metric collection option when creating a new task definition. For more information, see Creating an Amazon ECS task definition using the console.

The application metrics are exported to CloudWatch Logs with log group name `/aws/ecs/application/metrics` and the metrics can be viewed in the `ECS/AWSOTel/Application` namespace. Your application must be instrumented with the OpenTelemetry SDK. For more information, see Introduction to AWS Distro for OpenTelemetry in the AWS Distro for OpenTelemetry documentation.

Considerations

The following should be considered when using the Amazon ECS on Fargate integration with AWS Distro for OpenTelemetry to send application metrics to Amazon CloudWatch.

- This integration only sends your custom application metrics to CloudWatch. If you want task-level metrics, you can turn on Container Insights in the Amazon ECS cluster configuration. For more information, see Monitor Amazon ECS containers using Container Insights with enhanced observability.
- The AWS Distro for OpenTelemetry integration is supported for Amazon ECS workloads hosted on Fargate and Amazon ECS workloads hosted on Amazon EC2 instances. External instances aren't currently supported.
- CloudWatch supports a maximum of 30 dimensions per metric. By default, Amazon ECS defaults to including the `TaskARN`, `ClusterARN`, `LaunchType`, `TaskDefinitionFamily`, and `TaskDefinitionRevision` dimensions to the metrics. The remaining 25 dimensions can be defined by your application. If more than 30 dimensions are configured, CloudWatch can't display them. When this occurs, the application metrics will appear in the `ECS/AWSOTel/Application` CloudWatch metric namespace but without any dimensions. You can instrument your application to add additional dimensions. For more information, see Using CloudWatch metrics with AWS Distro for OpenTelemetry in the AWS Distro for OpenTelemetry documentation.

Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon CloudWatch

The Amazon ECS integration with AWS Distro for OpenTelemetry requires that you create a task IAM role and specify the role in your task definition. We recommend that the AWS Distro for OpenTelemetry sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The Amazon ECS console takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually and added to your task definition. For more information about the task execution IAM role, see Amazon ECS task execution IAM role.

> **Important**
>
> If you're also collecting application trace data using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see Identify Amazon ECS optimization opportunities using application trace data.
> If your application requires any additional permissions, you should add them to this policy. Each task definition may only specify one task IAM role. For example, if you are using a custom configuration file stored in Systems Manager, you should add the `ssm:GetParameters` permission to this IAM policy.

To create the service role for Elastic Container Service (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **Elastic Container Service**, and then choose the **Elastic Container Service Task** use case.
5. Choose **Next**.
6. In the **Add permissions** section, search for **AWSDistroOpenTelemetryPolicyForXray**, then select the policy.
7. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

    a. Open the **Set permissions boundary** section, and then choose **Use a permissions boundary to control the maximum role permissions**.

       IAM includes a list of the AWS managed and customer-managed policies in your account.

    b. Select the policy to use for the permissions boundary.

8. Choose **Next**.
9. Enter a role name or a role name suffix to help you identify the purpose of the role.

> **Important**
>
> When you name a role, note the following:
>
> - Role names must be unique within your AWS account, and can't be made unique by case.
>
>   For example, don't create roles named both **PRODROLE** and **prodrole**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.
> - You can't edit the name of the role after it's created because other entities might reference the role.

10. (Optional) For **Description**, enter a description for the role.
11. (Optional) To edit the use cases and permissions for the role, in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.
12. (Optional) To help identify, organize, or search for the role, add tags as key-value pairs. For more information about using tags in IAM, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.
13. Review the role, and then choose **Create role**.

Specifying the AWS Distro for OpenTelemetry sidecar in your task definition

The Amazon ECS console simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use metric collection** option. For more information, see [Creating an Amazon ECS task definition using the console](#).

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition manually. The following task definition example shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for Amazon CloudWatch integration.

```
{
```

```
  "family": "otel-using-cloudwatch",
  "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
  "containerDefinitions": [
     {
    "name": "aws-otel-emitter",
    "image": "application-image",
    "logConfiguration": {
     "logDriver": "awslogs",
     "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/aws-otel-emitter",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
     }
    },
    "dependsOn": [{
     "containerName": "aws-otel-collector",
     "condition": "START"
    }]
   },
   {
    "name": "aws-otel-collector",
    "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
    "essential": true,
    "command": [
     "--config=/etc/ecs/ecs-cloudwatch.yaml"
    ],
    "logConfiguration": {
     "logDriver": "awslogs",
     "options": {
      "awslogs-create-group": "True",
      "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
     }
    }
   }
  ],
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
   "FARGATE"
  ],
  "cpu": "1024",
```

```
  "memory": "3072"
}
```

Exporting application metrics to Amazon Managed Service for Prometheus

Amazon ECS supports exporting your task-level CPU, memory, network, and storage metrics and your custom application metrics to Amazon Managed Service for Prometheus. This is done by adding the AWS Distro for OpenTelemetry sidecar container to your task definition. The Amazon ECS console simplifies this process by adding the **Use metric collection** option when creating a new task definition. For more information, see Creating an Amazon ECS task definition using the console.

The metrics are exported to Amazon Managed Service for Prometheus and can be viewed using the Amazon Managed Grafana dashboard. Your application must be instrumented with either Prometheus libraries or with the OpenTelemetry SDK. For more information about instrumenting your application with the OpenTelemetry SDK, see Introduction to AWS Distro for OpenTelemetry in the AWS Distro for OpenTelemetry documentation.

When using the Prometheus libraries, your application must expose a `/metrics` endpoint which is used to scrape the metrics data. For more information about instrumenting your application with Prometheus libraries, see Prometheus client libraries in the Prometheus documentation.

Considerations

The following should be considered when using the Amazon ECS on Fargate integration with AWS Distro for OpenTelemetry to send application metrics to Amazon Managed Service for Prometheus.

- The AWS Distro for OpenTelemetry integration is supported for Amazon ECS workloads hosted on Fargate and Amazon ECS workloads hosted on Amazon EC2 instances. External instances aren't supported currently.
- By default, AWS Distro for OpenTelemetry includes all available task-level dimensions for your application metrics when exporting to Amazon Managed Service for Prometheus. You can also instrument your application to add additional dimensions. For more information, see Getting Started with Prometheus Remote Write Exporter for Amazon Managed Service for Prometheus in the AWS Distro for OpenTelemetry documentation.

Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus

The Amazon ECS integration with Amazon Managed Service for Prometheus using the AWS Distro for OpenTelemetry sidecar requires that you create a task IAM role and specify the role in your task definition. This task IAM role must be created manually using the steps below prior to registering your task definition.

We recommend that the AWS Distro for OpenTelemetry sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The Amazon ECS console takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually. For more information about creating a task execution IAM role, see Amazon ECS task execution IAM role.

> **Important**
>
> If you're also collecting application trace data using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see Identify Amazon ECS optimization opportunities using application trace data.

To create the service role for Elastic Container Service (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **Elastic Container Service**, and then choose the **Elastic Container Service Task** use case.
5. Choose **Next**.
6. In the **Add permissions** section, search for **AmazonPrometheusRemoteWriteAccess**, then select the policy.
7. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

a. Open the **Set permissions boundary** section, and then choose **Use a permissions boundary to control the maximum role permissions**.

   IAM includes a list of the AWS managed and customer-managed policies in your account.

b. Select the policy to use for the permissions boundary.

8. Choose **Next**.
9. Enter a role name or a role name suffix to help you identify the purpose of the role.

> **Important**
>
> When you name a role, note the following:
>
> - Role names must be unique within your AWS account, and can't be made unique by case.
>
>   For example, don't create roles named both **`PRODROLE`** and **`prodrole`**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.
> - You can't edit the name of the role after it's created because other entities might reference the role.

10. (Optional) For **Description**, enter a description for the role.
11. (Optional) To edit the use cases and permissions for the role, in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.
12. (Optional) To help identify, organize, or search for the role, add tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources in the *IAM User Guide*.
13. Review the role, and then choose **Create role**.

Specifying the AWS Distro for OpenTelemetry sidecar in your task definition

The Amazon ECS console simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use metric collection** option. For more information, see Creating an Amazon ECS task definition using the console.

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition manually. The following task definition example shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for Amazon Managed Service for Prometheus integration.

```
{
 "family": "otel-using-cloudwatch",
 "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",
 "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
 "containerDefinitions": [{
   "name": "aws-otel-emitter",
   "image": "application-image",
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "true",
     "awslogs-group": "/ecs/aws-otel-emitter",
     "awslogs-region": "aws-region",
     "awslogs-stream-prefix": "ecs"
    }
   },
   "dependsOn": [{
    "containerName": "aws-otel-collector",
    "condition": "START"
   }]
  },
  {
   "name": "aws-otel-collector",
   "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.30.0",
   "essential": true,
   "command": [
    "--config=/etc/ecs/ecs-amp.yaml"
   ],
   "environment": [{
    "name": "AWS_PROMETHEUS_ENDPOINT",
    "value": "https://aps-workspaces.aws-region.amazonaws.com/workspaces/
ws-a1b2c3d4-5678-90ab-cdef-EXAMPLE11111/api/v1/remote_write"
   }],
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-create-group": "True",
     "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
```

```
        "awslogs-region": "aws-region",
        "awslogs-stream-prefix": "ecs"
      }
    }
  }
 ],
 "networkMode": "awsvpc",
 "requiresCompatibilities": [
  "FARGATE"
 ],
 "cpu": "1024",
 "memory": "3072"
}
```

Log Amazon ECS API calls using AWS CloudTrail

Amazon Elastic Container Service is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures all API calls for Amazon ECS as events. The calls captured include calls from the Amazon ECS console and code calls to the Amazon ECS API operations. Using the information collected by CloudTrail, you can determine the request that was made to Amazon ECS, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

Amazon ECS management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

Amazon Elastic Container Service logs all Amazon ECS control plane operations as management events. For example, calls to the `CreateService`, `RunTask` and `DeleteCluster` sections generate entries in the CloudTrail log files. For a list of the Amazon Elastic Container Service control plane operations that Amazon ECS logs to CloudTrail, see the [Amazon Elastic Container Service API Reference](#).

Amazon ECS event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

The following example shows a CloudTrail event that demonstrates the `CreateService` action.

> **Note**
>
> This example has been formatted for improved readability. In a CloudTrail log file, all entries and events are concatenated into a single line. In addition, this example has been limited to a single Amazon ECS entry. In a real CloudTrail log file, you see entries and events from multiple AWS services.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:account_name",
        "arn": "arn:aws:sts::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-06-20T18:32:25Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Mary_Major"
```

```
            }
        }
    },
    "eventTime": "2018-06-20T19:04:36Z",
    "eventSource": "ecs.amazonaws.com",
    "eventName": "CreateCluster",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "console.amazonaws.com",
    "requestParameters": {
        "clusterName": "default"
    },
    "responseElements": {
        "cluster": {
            "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/default",
            "pendingTasksCount": 0,
            "registeredContainerInstancesCount": 0,
            "status": "ACTIVE",
            "runningTasksCount": 0,
            "statistics": [],
            "clusterName": "default",
            "activeServicesCount": 0
        }
    },
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Viewing CloudWatch Logs Live Tail for Amazon ECS services and tasks

CloudWatch Logs Live Tail helps you quickly troubleshoot incidents by viewing a streaming list of new log events as they are ingested. You can view task and service events in the Amazon ECS console. This view provides a cohesive view of your task and service health.

Each task in an Amazon ECS service has dedicated log streams for each container. If a service scales up, each task instance has its own set of log streams. The naming convention for log streams follows the pattern:

```
log-group-prefix/container-name/task-id
```

A single task writes to the same log streams during its lifetime. The log stream contains messages from that task's containers and also any output from your application code. Every message is timestamped, including your custom logs.

Note

Live Tail sessions incur costs by session usage time, per minute. For more information about pricing, see Amazon CloudWatch Pricing.

Required permissions

The following permissions are required for the console IAM user to start and stop CloudWatch Logs Live Tail sessions:

- `logs:DescribeLogGroups`
- `logs:StartLiveTail`
- `logs:StopLiveTail`

Procedure

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. Determine the resource you want to view

Resource	Steps	
Tasks	a. On the **Clusters** page, choose the cluster. The cluster details page displays.	

Resource	Steps	
	b. Choose the **Tasks** tab. The task details page displays. c. Choose the **Logs** tab.	
Services	a. On the **Clusters** page, choose the cluster. The cluster details page displays. b. Choose the service. The service details page displays. c. Choose the **Logs** tab.	

3. Choose **CloudWatch Logs Live Tail**, and then choose **Start**.
4. (Optional) To filter the streams, under **Filter**, for **Select log groups**, choose the log group.

AWS CLI references

You can also use the AWS CLI to start Live Tail sessions for CloudWatch Logs. The following resources provide detailed information about using the AWS CLI with Live Tail:

- start-live-tail command reference - Complete command syntax, parameters, and examples for the `aws logs start-live-tail` command.
- CloudWatch Logs Live Tail user guide - Comprehensive guide including AWS CLI usage with both print-only and interactive modes.
- StartLiveTail SDK examples - Programmatic examples for using the StartLiveTail API with various AWS SDKs.

Monitor workloads using Amazon ECS metadata

You can use the task and container metadata to troubleshoot your workloads and to make configuration changes based on the runtime environment.

Metadata includes the following categories:

- Task-level attributes that provide information about where the task is running.
- Container-level attributes that provide the Docker ID, name, and image details.

  This provides visibility into the container.
- Network settings such as IP addresses, subnets, and network mode.

  This helps with network configuration and troubleshooting.
- Task status and health

  This lets you know if the tasks are running.

You can view metadata by any of the following methods:

- Container metadata file

  Beginning with version 1.15.0 of the Amazon ECS container agent, various container metadata is available within your containers or the host container instance. By enabling this feature, you can query the information about a task, container, and container instance from within the container or the host container instance. The metadata file is created on the host instance and mounted in the container as a Docker volume and therefore is not available when a task is hosted on AWS Fargate.
- Task metadata endpoint

  The Amazon ECS container agent injects an environment variable into each container, referred to as the *task metadata endpoint* which provides various task metadata and [Docker stats](https://docs.docker.com/engine/api/v1.30/#operation/ContainerStats) to the container.
- Container introspection

  The Amazon ECS container agent provides an API operation for gathering details about the container instance on which the agent is running and the associated tasks running on that instance.

Amazon ECS environment variables

Environment variables control a task's behavior such as the capacity used to run the task. Amazon ECS sets the following environment variables for your tasks:

- `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` - The full HTTP URL endpoint for the SDK to use when making a request for credentials. This includes both the scheme and the host. For more information, see Container credential provider in the *AWS SDKs Reference Guide*.
- `ECS_CONTAINER_METADATA_URI_V4` - The address of the task metadata version 4. For more information, see Amazon ECS task metadata endpoint version 4.
- `ECS_CONTAINER_METADATA_URI` - The address of the task metadata version 3. For more information, see Amazon ECS task metadata endpoint version 3.
- `ECS_AGENT_URI` - The base address for different endpoints supported by Fargate. For more information, see:
  - Amazon ECS task scale-in protection endpoint
  - Amazon ECS fault injection endpoints
- `AWS_EXECUTION_ENV` - The information about the launch type the task runs on.
  - For Fargate, Amazon ECS sets this to `AWS_ECS_FARGATE`.
  - For EC2, Amazon ECS sets this to `AWS_ECS_EC2`.
- `AWS_DEFAULT_REGION` – The default AWS Region for your AWS account. This is the default Region where your task runs.
- `AWS_REGION` – The Region where the task runs. If defined, this value overrides the `AWS_DEFAULT_REGION`.

You can view the environment variables in the task metadata. For more information, see one of the following topics:

- Fargate - Amazon ECS task metadata available for tasks on Fargate
- EC2 - Task metadata available for Amazon ECS tasks on EC2

Amazon ECS container metadata file

Beginning with version 1.15.0 of the Amazon ECS container agent, various container metadata is available within your containers or the host container instance. By enabling this feature, you can query the information about a task, container, and container instance from within the container or the host container instance. The metadata file is created on the host instance and mounted in the container as a Docker volume and therefore is not available when a task is hosted on AWS Fargate.

The container metadata file is cleaned up on the host instance when the container is cleaned up. You can adjust when this happens with the `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` container agent variable. For more information, see Automatic Amazon ECS task and image cleanup.

Topics

- Container metadata file locations
- Turning on Amazon ECS container metadata
- Amazon ECS container metadata file format

Container metadata file locations

By default, the container metadata file is written to the following host and container paths.

- **For Linux instances:**
  - Host path: `/var/lib/ecs/data/metadata/cluster_name/task_id/container_name/ecs-container-metadata.json`

    > **Note**
    >
    > The Linux host path assumes that the default data directory mount path (`/var/lib/ecs/data`) is used when the agent is started. If you are not using an Amazon ECS-optimized AMI (or the `ecs-init` package to start and maintain the container agent), be sure to set the `ECS_HOST_DATA_DIR` agent configuration variable to the host path where the container agent's state file is located. For more information, see Amazon ECS container agent configuration.

  - Container path: `/opt/ecs/metadata/random_ID/ecs-container-metadata.json`
- **For Windows instances:**
  - Host path: `C:\ProgramData\Amazon\ECS\data\metadata\task_id\container_name\ecs-container-metadata.json`

- Container path: `C:\ProgramData\Amazon\ECS\metadata\random_ID\ecs-container-metadata.json`

However, for easy access, the container metadata file location is set to the `ECS_CONTAINER_METADATA_FILE` environment variable inside the container. You can read the file contents from inside the container with the following command:

- **For Linux instances:**

```
cat $ECS_CONTAINER_METADATA_FILE
```

- **For Windows instances (PowerShell):**

```
Get-Content -path $env:ECS_CONTAINER_METADATA_FILE
```

Turning on Amazon ECS container metadata

You can turn on container metadata at the container instance level by setting the `ECS_ENABLE_CONTAINER_METADATA` container agent variable to `true`. You can set this variable in the `/etc/ecs/ecs.config` configuration file and restart the agent. You can also set it as a Docker environment variable at runtime when the agent container is started. For more information, see Amazon ECS container agent configuration.

If the `ECS_ENABLE_CONTAINER_METADATA` is set to `true` when the agent starts, metadata files are created for any containers created from that point forward. The Amazon ECS container agent cannot create metadata files for containers that were created before the `ECS_ENABLE_CONTAINER_METADATA` container agent variable was set to `true`. To ensure that all containers receive metadata files, you should set this agent variable at container instance launch. The following is an example user data script that will set this variable as well as register your container instance with your cluster.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_ENABLE_CONTAINER_METADATA=true
EOF
```

Amazon ECS container metadata file format

The following information is stored in the container metadata JSON file.

`Cluster`

The name of the cluster that the container's task is running on.

`ContainerInstanceARN`

The full Amazon Resource Name (ARN) of the host container instance.

`TaskARN`

The full Amazon Resource Name (ARN) of the task that the container belongs to.

`TaskDefinitionFamily`

The name of the task definition family the container is using.

`TaskDefinitionRevision`

The task definition revision the container is using.

`ContainerID`

The Docker container ID (and not the Amazon ECS container ID) for the container.

`ContainerName`

The container name from the Amazon ECS task definition for the container.

`DockerContainerName`

The container name that the Docker daemon uses for the container (for example, the name that shows up in **docker ps** command output).

`ImageID`

The SHA digest for the Docker image used to start the container.

`ImageName`

The image name and tag for the Docker image used to start the container.

`PortMappings`

Any port mappings associated with the container.

`ContainerPort`

The port on the container that is exposed.

`HostPort`

The port on the host container instance that is exposed.

`BindIp`

The bind IP address that is assigned to the container by Docker. This IP address is only applied with the `bridge` network mode, and it is only accessible from the container instance.

`Protocol`

The network protocol used for the port mapping.

`Networks`

The network mode and IP address for the container.

`NetworkMode`

The network mode for the task to which the container belongs.

`IPv4Addresses`

The IP addresses associated with the container.

> **Important**
>
> If your task is using the `awsvpc` network mode, the IP address of the container will not be returned. In this case, you can retrieve the IP address by reading the /etc/hosts file with the following command:
>
> ```
> tail -1 /etc/hosts | awk '{print $1}'
> ```

`MetadataFileStatus`

The status of the metadata file. When the status is `READY`, the metadata file is current and complete. If the file is not ready yet (for example, the moment the task is started), a truncated version of the file format is available. To avoid a likely race condition where the container has started, but the metadata has not yet been written, you can parse the metadata file and wait for this parameter to be set to READY before depending on the metadata. This is usually available in less than 1 second from when the container starts.

`AvailabilityZone`

The Availability Zone the host container instance resides in.

`HostPrivateIPv4Address`

The private IP address for the task the container belongs to.

`HostPublicIPv4Address`

The public IP address for the task the container belongs to.

Example Amazon ECS container metadata file (READY)

The following example shows a container metadata file in the READY status.


```
{
    "Cluster":"arn:aws:ecs:us-east-1:123456789012:cluster/MyCluster",
    "TaskARN":"arn:aws:ecs:us-east-1:123456789012:task/MyCluster/
b593651c4d6b44a6b2b583f45c957e15",
    "Family":"curltest-container",
    "Revision":"2",
    "DesiredStatus":"RUNNING",
    "KnownStatus":"RUNNING",
    "Limits":
        {
            "CPU":0.25,
            "Memory":512
        },
    "PullStartedAt":"2025-01-17T20:56:17.394610044Z",
    "PullStoppedAt":"2025-01-17T20:56:25.282708213Z",
    "AvailabilityZone":"us-east-1b",
    "LaunchType":"FARGATE",
    "Containers":[
        {
            "DockerId":"b593651c4d6b44a6b2b583f45c957e15-3356213583",
            "Name":"curltest","DockerName":"curltest",
            "Image":"public.ecr.aws/amazonlinux/amazonlinux:latest",

 "ImageID":"sha256:7f371357694782356b65c7fd60dd1ca124c47bd5ed1b1ffe7c0e17f562898367",
            "Labels":
                {
```

```
                    "com.amazonaws.ecs.cluster":"arn:aws:ecs:us-
east-1:123456789012:cluster/MyCluster",
                    "com.amazonaws.ecs.container-name":"curltest",
                    "com.amazonaws.ecs.task-arn":"arn:aws:ecs:us-
east-1:123456789012:task/MyCluster/b593651c4d6b44a6b2b583f45c957e15",
                    "com.amazonaws.ecs.task-definition-family":"curltest-
container","com.amazonaws.ecs.task-definition-version":"2"
               },
            "DesiredStatus":"RUNNING",
            "KnownStatus":"RUNNING",
            "Limits":
                {
                    "CPU":2
                },
            "CreatedAt":"2025-01-17T20:56:26.180347056Z",
            "StartedAt":"2025-01-17T20:56:26.180347056Z",
            "Type":"NORMAL",
            "LogDriver":"awslogs",
            "LogOptions":
                {
                    "awslogs-create-group":"true",
                    "awslogs-group":"/ecs/curltest-container",
                    "awslogs-region":"us-east-1",
                    "awslogs-stream":"ecs/curltest/b593651c4d6b44a6b2b583f45c957e15"
                       },
            "ContainerARN":"arn:aws:ecs:us-east-1:123456789012:container/MyCluster/
b593651c4d6b44a6b2b583f45c957e15/934575e8-5bdb-478f-b763-2341a85b690e",
            "Networks":[
                {
                    "NetworkMode":"awsvpc",
                    "IPv4Addresses":["10.0.1.58"]
                }
            ],
            "Snapshotter":"overlayfs"
        }
    ],
    "ClockDrift":
        {
            "ClockErrorBound":0.487801,"ReferenceTimestamp":"2025-01-17T20:56:02Z",
            "ClockSynchronizationStatus":"SYNCHRONIZED"
        },
    "FaultInjectionEnabled":false
}
```

Example Incomplete Amazon ECS container metadata file (not yet READY)

The following example shows a container metadata file that has not yet reached the READY status. The information in the file is limited to a few parameters that are known from the task definition. The container metadata file should be ready within 1 second after the container starts.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/
default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/
d90675f8-1a98-444b-805b-3d9cabb6fcd4",
    "ContainerName": "metadata"
}
```

Task metadata available for Amazon ECS tasks on EC2

The Amazon ECS container agent provides a method to retrieve various task metadata and Docker stats. This is referred to as the task metadata endpoint. The following versions are available:

- Task metadata endpoint version 4 – Provides a variety of metadata and Docker stats to containers. Can also provide network rate data. Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version `1.39.0` of the Amazon ECS container agent. For Amazon EC2 Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version `1.54.0`. For more information, see Amazon ECS task metadata endpoint version 4.
- Task metadata endpoint version 3 – Provides a variety of metadata and Docker stats to containers. Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version `1.21.0` of the Amazon ECS container agent. For Amazon EC2 Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version `1.54.0`. For more information, see Amazon ECS task metadata endpoint version 3.
- Task metadata endpoint version 2 – Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version `1.17.0` of the Amazon ECS container agent. For Amazon EC2 Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version `1.54.0`. For more information, see Amazon ECS task metadata endpoint version 2.

If your Amazon ECS task is hosted on Amazon EC2, you can also access task host metadata using the Instance Metadata Service (IMDS) endpoint. The following command, when run from within the instance hosting the task, lists the ID of the host instance.

```
curl http://169.254.169.254/latest/meta-data/instance-id
```

The information you can obtain from the endpoint is divided into categories such as *instance-id*. For more information about the different categories of host instance metadata you can obtain using the endpoint, see Instance metadata categories .

Amazon ECS task metadata endpoint version 4

The Amazon ECS container agent injects an environment variable into each container, referred to as the *task metadata endpoint* which provides various task metadata and Docker stats to the container.

The task metadata and network rate stats are sent to CloudWatch Container Insights and can be viewed in the AWS Management Console. For more information, see Monitor Amazon ECS containers using Container Insights with enhanced observability.

> **Note**
>
> Amazon ECS provides earlier versions of the task metadata endpoint. To avoid the need to create new task metadata endpoint versions in the future, additional metadata may be added to the version 4 output. We will not remove any existing metadata or change the metadata field names.

The environment variable is injected by default into the containers of Amazon ECS tasks launched on Amazon EC2 Linux instances that are running at least version `1.39.0` of the Amazon ECS container agent. For Amazon EC2 Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version `1.54.0`. For more information, see Amazon ECS Linux container instance management.

> **Note**
>
> You can add support for this feature on Amazon EC2 instances using older versions of the Amazon ECS container agent by updating the agent to the latest version. For more information, see Updating the Amazon ECS container agent.

For task metadata example output, see Amazon ECS task metadata v4 examples.

Task metadata endpoint version 4 paths

The following task metadata endpoint paths are available to containers.

`${ECS_CONTAINER_METADATA_URI_V4}`

This path returns metadata for the container.

`${ECS_CONTAINER_METADATA_URI_V4}/task`

This path returns metadata for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see Amazon ECS task metadata V4 JSON response.

`${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`

This path returns the metadata for the task included in the `/task` endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API. Any errors received when retrieving the tag metadata will be included in the `Errors` field in the response.

> **Note**
>
> The `Errors` field is only in the response for tasks hosted on Amazon EC2 Linux instances running at least version `1.50.0` of the container agent. For Amazon EC2 Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version `1.54.0`
> This endpoint requires the `ecs.ListTagsForResource` permission.

> **Important**
>
> When using the `${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags` endpoint, be aware that each call makes up to two API requests to `ecs:ListTagsForResource` (one for container instance tags and one for task tags) and that any sidecar containers in the task may make these calls on your behalf. Frequent endpoint calls can result in API throttling.
> Consider implementing caching or batching strategies to reduce the frequency of calls, especially in high-traffic applications, and debugging API throttling issues using AWS CloudTrail. For information about throttling limits for the `ListTagsForResource` API, see Request throttling for the Amazon ECS API in the *Amazon Elastic Container Service API Reference*. For more information about debugging Amazon ECS API calls using AWS CloudTrail, see Log Amazon ECS API calls using AWS CloudTrail.

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

This path returns Docker stats for the specific container. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

For Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 Linux instances running at least version `1.43.0` of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

This path returns Docker stats for all of the containers associated with the task. This can be used by sidecar containers to extract network metrics. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

For Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 Linux instances running at least version `1.43.0` of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

Amazon ECS task metadata V4 JSON response

The following information is returned from the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON response. This includes metadata associated with the task in addition to the metadata for each container within the task.

`Cluster`

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

`ServiceName`

The name of the service to which the task belongs. ServiceName will appear for Amazon EC2 and Amazon ECS Anywhere container instances if the task is associated with a service.

> **Note**
>
> The `ServiceName` metadata is only included when using Amazon ECS container agent version `1.63.1` or later.

`VPCID`

The VPC ID of the Amazon EC2 container instance. This field only appears for Amazon EC2 instances.

> **Note**
>
> The `VPCID` metadata is only included when using Amazon ECS container agent version `1.63.1` or later.

`TaskARN`

The Amazon Resource Name (ARN) of the task to which the container belongs.

`Family`

The family of the Amazon ECS task definition for the task.

`Revision`

The revision of the Amazon ECS task definition for the task.

`DesiredStatus`

The desired status for the task from Amazon ECS.

`KnownStatus`

The known status for the task from Amazon ECS.

`Limits`

The resource limits specified at the task level, such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

`PullStartedAt`

The timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

> **Note**
>
> The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 (Windows).

`LaunchType`

The launch type the task is using. When using cluster capacity providers, this indicates whether the task is using Fargate or EC2 infrastructure.

> **Note**
>
> This `LaunchType` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later (Linux) or 1.0.0 or later (Windows).

`Containers`

A list of container metadata for each container associated with the task.

`DockerId`

The Docker ID for the container.

When you use Fargate, the id is a 32-digit hex followed by a 10 digit number.

`Name`

The name of the container as specified in the task definition.

`DockerName`

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

`Image`

The image for the container.

`ImageID`

The SHA-256 digest of the image manifest. This is the digest that can be used to pull the image using the format `repository-url/image@sha256:digest`.

`Ports`

Any ports exposed for the container. This parameter is omitted if there are no exposed ports.

`Labels`

Any labels applied to the container. This parameter is omitted if there are no labels applied.

`DesiredStatus`

The desired status for the container from Amazon ECS.

`KnownStatus`

The known status for the container from Amazon ECS.

`ExitCode`

The exit code for the container. This parameter is omitted if the container has not exited.

`Limits`

The resource limits specified at the container level, such as CPU (expressed in CPU units) and memory. This parameter is omitted if no resource limits are defined.

`CreatedAt`

The time stamp for when the container was created. This parameter is omitted if the container has not been created yet.

`StartedAt`

The time stamp for when the container started. This parameter is omitted if the container has not started yet.

`FinishedAt`

The time stamp for when the container stopped. This parameter is omitted if the container has not stopped yet.

`Type`

The type of the container. Containers that are specified in your task definition are of type `NORMAL`. You can ignore other container types, which are used for internal task resource provisioning by the Amazon ECS container agent.

`LogDriver`

The log driver the container is using.

> **Note**
>
> This `LogDriver` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later.

`LogOptions`

The log driver options defined for the container.

> **Note**
>
> This `LogOptions` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later.

`ContainerARN`

The Amazon Resource Name (ARN) of the container.

> **Note**
>
> This `ContainerARN` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later.

`Networks`

The network information for the container, such as the network mode and IP address. This parameter is omitted if no network information is defined.

`RestartCount`

The number of times the container has been restarted.

> **Note**
>
> The `RestartCount` metadata is included only if a restart policy is enabled for the container. For more information, see [Restart individual containers in Amazon ECS tasks with container restart policies](#).

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Amazon ECS task metadata v4 examples

The following examples show example outputs from each of the task metadata endpoints.

Example container metadata response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}` endpoint you are returned only metadata about the container itself. The following is an example output from a task that runs as part of a service (MyService).

```
{
    "DockerId": "ea32192c8553fbff06c9340478a2ff089b2bb5646fb718b4ee206641c9086d66",
    "Name": "curl",
    "DockerName": "ecs-curltest-24-curl-cca48e8dcadd97805600",
```

```
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/8f03e41243824aea923aca126495f665",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "24"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:15:07.620912337Z",
    "StartedAt": "2020-10-02T00:15:08.062559351Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/8f03e41243824aea923aca126495f665"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/0206b271-
b33f-47ab-86c6-a0ba208a70a9",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.100"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:9e:32:c7:48:85",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-100.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}
```

Example task metadata response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/task` endpoint you are returned metadata about the task the container is part of in addition to the metadata for each container within the task. The following is an example output.


```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
```

```
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
```

```
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

Example task with tags metadata response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags` endpoint you are returned metadata about the task, including the task and container instance tags. The following is an example output.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "TaskTags": {
```

```
        "tag-use": "task-metadata-endpoint-test"
    },
    "ContainerInstanceTags":{
        "tag_key":"tag_value"
    },
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
```

```
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
```

```
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

Example task with tags with an error metadata response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags` endpoint you are returned metadata about the task, including the task and container instance tags. If there is an error retrieving the tagging data, the error is returned in the response. The following is an example output for when the IAM role associated with the container instance doesn't have the `ecs:ListTagsForResource` permission allowed.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "ServiceName": "MyService",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "VPCID": "vpc-1234567890abcdef0",
    "Errors": [
        {
            "ErrorField": "ContainerInstanceTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-
role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform:
 ecs:ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:container-
instance/default/2dd1b186f39845a584488d2ef155c131",
            "StatusCode": 400,
            "RequestId": "cd597ef0-272b-4643-9bd2-1de469870fa6",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:container-instance/
default/2dd1b186f39845a584488d2ef155c131"
        },
        {
```

```
            "ErrorField": "TaskTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-
role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform:
 ecs:ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:task/
default/9ef30e4b7aa44d0db562749cff4983f3",
            "StatusCode": 400,
            "RequestId": "862c5986-6cd2-4aa6-87cc-70be395531e1",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/9ef30e4b7aa44d0db562749cff4983f3"
        }
    ],
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId":
 "598cba581fe3f939459eaba1e071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal~ecs~pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2020-10-02T00:43:05.602352471Z",
            "StartedAt": "2020-10-02T00:43:06.076707576Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
```

```
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        },
        {
            "DockerId":
 "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
            "Name": "curl",
            "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
            "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
            "ImageID":
 "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 10,
                "Memory": 128
            },
            "CreatedAt": "2020-10-02T00:43:06.326590752Z",
            "StartedAt": "2020-10-02T00:43:06.767535449Z",
            "Type": "NORMAL",
            "LogDriver": "awslogs",
            "LogOptions": {
                "awslogs-create-group": "true",
                "awslogs-group": "/ecs/metadata",
                "awslogs-region": "us-west-2",
                "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
            },
            "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
            "Networks": [
                {
```

```
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.61"
                    ],
                    "AttachmentIndex": 0,
                    "MACAddress": "0e:10:e2:01:bd:91",
                    "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                    "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": "10.0.2.1/24"
                }
            ]
        }
    ]
}
```

Example container stats response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/stats` endpoint you are returned network metrics for the container. For Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 instances running at least version `1.43.0` of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

The following is an example output from an Amazon ECS task on Amazon EC2 that uses the `bridge` network mode.

```
{
    "read": "2020-10-02T00:51:13.410254284Z",
    "preread": "2020-10-02T00:51:12.406202398Z",
    "pids_stats": {
        "current": 3
    },
    "blkio_stats": {
        "io_service_bytes_recursive": [

        ],
        "io_serviced_recursive": [

        ],
        "io_queue_recursive": [

        ],
        "io_service_time_recursive": [
```

```
        ],
        "io_wait_time_recursive": [

        ],
        "io_merged_recursive": [

        ],
        "io_time_recursive": [

        ],
        "sectors_recursive": [

        ]
    },
    "num_procs": 0,
    "storage_stats": {

    },
    "cpu_stats": {
        "cpu_usage": {
            "total_usage": 360968065,
            "percpu_usage": [
                182359190,
                178608875
            ],
            "usage_in_kernelmode": 40000000,
            "usage_in_usermode": 290000000
        },
        "system_cpu_usage": 13939680000000,
        "online_cpus": 2,
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
            "throttled_time": 0
        }
    },
    "precpu_stats": {
        "cpu_usage": {
            "total_usage": 360968065,
            "percpu_usage": [
                182359190,
                178608875
            ],
```

```
            "usage_in_kernelmode": 40000000,
            "usage_in_usermode": 290000000
        },
        "system_cpu_usage": 13937670000000,
        "online_cpus": 2,
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
            "throttled_time": 0
        }
    },
    "memory_stats": {
        "usage": 1806336,
        "max_usage": 6299648,
        "stats": {
            "active_anon": 606208,
            "active_file": 0,
            "cache": 0,
            "dirty": 0,
            "hierarchical_memory_limit": 134217728,
            "hierarchical_memsw_limit": 268435456,
            "inactive_anon": 0,
            "inactive_file": 0,
            "mapped_file": 0,
            "pgfault": 4185,
            "pgmajfault": 0,
            "pgpgin": 2926,
            "pgpgout": 2778,
            "rss": 606208,
            "rss_huge": 0,
            "total_active_anon": 606208,
            "total_active_file": 0,
            "total_cache": 0,
            "total_dirty": 0,
            "total_inactive_anon": 0,
            "total_inactive_file": 0,
            "total_mapped_file": 0,
            "total_pgfault": 4185,
            "total_pgmajfault": 0,
            "total_pgpgin": 2926,
            "total_pgpgout": 2778,
            "total_rss": 606208,
            "total_rss_huge": 0,
            "total_unevictable": 0,
```

```
            "total_writeback": 0,
            "unevictable": 0,
            "writeback": 0
        },
        "limit": 134217728
    },
    "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
    "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
    "networks": {
        "eth0": {
            "rx_bytes": 84,
            "rx_packets": 2,
            "rx_errors": 0,
            "rx_dropped": 0,
            "tx_bytes": 84,
            "tx_packets": 2,
            "tx_errors": 0,
            "tx_dropped": 0
        }
    },
    "network_rate_stats": {
        "rx_bytes_per_sec": 0,
        "tx_bytes_per_sec": 0
    }
}
```

Example task stats response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/task/stats` endpoint you are returned network metrics about the task the container is part of. The following is an example output.

```
{
    "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854": {
        "read": "2020-10-02T00:51:32.51467703Z",
        "preread": "2020-10-02T00:51:31.50860463Z",
        "pids_stats": {
            "current": 1
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [

            ],
```

```
            "io_serviced_recursive": [

            ],
            "io_queue_recursive": [

            ],
            "io_service_time_recursive": [

            ],
            "io_wait_time_recursive": [

            ],
            "io_merged_recursive": [

            ],
            "io_time_recursive": [

            ],
            "sectors_recursive": [

            ]
        },
        "num_procs": 0,
        "storage_stats": {

        },
        "cpu_stats": {
            "cpu_usage": {
                "total_usage": 177232665,
                "percpu_usage": [
                    13376224,
                    163856441
                ],
                "usage_in_kernelmode": 0,
                "usage_in_usermode": 160000000
            },
            "system_cpu_usage": 13977820000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
```

```
        "precpu_stats": {
            "cpu_usage": {
                "total_usage": 177232665,
                "percpu_usage": [
                    13376224,
                    163856441
                ],
                "usage_in_kernelmode": 0,
                "usage_in_usermode": 160000000
            },
            "system_cpu_usage": 13975800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "memory_stats": {
            "usage": 532480,
            "max_usage": 6279168,
            "stats": {
                "active_anon": 40960,
                "active_file": 0,
                "cache": 0,
                "dirty": 0,
                "hierarchical_memory_limit": 9223372036854771712,
                "hierarchical_memsw_limit": 9223372036854771712,
                "inactive_anon": 0,
                "inactive_file": 0,
                "mapped_file": 0,
                "pgfault": 2033,
                "pgmajfault": 0,
                "pgpgin": 1734,
                "pgpgout": 1724,
                "rss": 40960,
                "rss_huge": 0,
                "total_active_anon": 40960,
                "total_active_file": 0,
                "total_cache": 0,
                "total_dirty": 0,
                "total_inactive_anon": 0,
                "total_inactive_file": 0,
                "total_mapped_file": 0,
```

```
                "total_pgfault": 2033,
                "total_pgmajfault": 0,
                "total_pgpgin": 1734,
                "total_pgpgout": 1724,
                "total_rss": 40960,
                "total_rss_huge": 0,
                "total_unevictable": 0,
                "total_writeback": 0,
                "unevictable": 0,
                "writeback": 0
            },
            "limit": 4073377792
        },
        "name": "/ecs-curltest-26-internalecspause-a6bcc3dbadfacfe85300",
        "id": "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854",
        "networks": {
            "eth0": {
                "rx_bytes": 84,
                "rx_packets": 2,
                "rx_errors": 0,
                "rx_dropped": 0,
                "tx_bytes": 84,
                "tx_packets": 2,
                "tx_errors": 0,
                "tx_dropped": 0
            }
        },
        "network_rate_stats": {
            "rx_bytes_per_sec": 0,
            "tx_bytes_per_sec": 0
        }
    },
    "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af": {
        "read": "2020-10-02T00:51:32.512771349Z",
        "preread": "2020-10-02T00:51:31.510597736Z",
        "pids_stats": {
            "current": 3
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [

            ],
            "io_serviced_recursive": [
```

```
            ],
            "io_queue_recursive": [

            ],
            "io_service_time_recursive": [

            ],
            "io_wait_time_recursive": [

            ],
            "io_merged_recursive": [

            ],
            "io_time_recursive": [

            ],
            "sectors_recursive": [

            ]
        },
        "num_procs": 0,
        "storage_stats": {

        },
        "cpu_stats": {
            "cpu_usage": {
                "total_usage": 379075681,
                "percpu_usage": [
                    191355275,
                    187720406
                ],
                "usage_in_kernelmode": 60000000,
                "usage_in_usermode": 310000000
            },
            "system_cpu_usage": 13977800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "precpu_stats": {
            "cpu_usage": {
```

```
                "total_usage": 378825197,
                "percpu_usage": [
                    191104791,
                    187720406
                ],
                "usage_in_kernelmode": 60000000,
                "usage_in_usermode": 310000000
            },
            "system_cpu_usage": 13975800000000,
            "online_cpus": 2,
            "throttling_data": {
                "periods": 0,
                "throttled_periods": 0,
                "throttled_time": 0
            }
        },
        "memory_stats": {
            "usage": 1814528,
            "max_usage": 6299648,
            "stats": {
                "active_anon": 606208,
                "active_file": 0,
                "cache": 0,
                "dirty": 0,
                "hierarchical_memory_limit": 134217728,
                "hierarchical_memsw_limit": 268435456,
                "inactive_anon": 0,
                "inactive_file": 0,
                "mapped_file": 0,
                "pgfault": 5377,
                "pgmajfault": 0,
                "pgpgin": 3613,
                "pgpgout": 3465,
                "rss": 606208,
                "rss_huge": 0,
                "total_active_anon": 606208,
                "total_active_file": 0,
                "total_cache": 0,
                "total_dirty": 0,
                "total_inactive_anon": 0,
                "total_inactive_file": 0,
                "total_mapped_file": 0,
                "total_pgfault": 5377,
                "total_pgmajfault": 0,
```

```
                "total_pgpgin": 3613,
                "total_pgpgout": 3465,
                "total_rss": 606208,
                "total_rss_huge": 0,
                "total_unevictable": 0,
                "total_writeback": 0,
                "unevictable": 0,
                "writeback": 0
            },
            "limit": 134217728
        },
        "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
        "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
        "networks": {
            "eth0": {
                "rx_bytes": 84,
                "rx_packets": 2,
                "rx_errors": 0,
                "rx_dropped": 0,
                "tx_bytes": 84,
                "tx_packets": 2,
                "tx_errors": 0,
                "tx_dropped": 0
            }
        },
        "network_rate_stats": {
            "rx_bytes_per_sec": 0,
            "tx_bytes_per_sec": 0
        }
    }
}
```


Amazon ECS task metadata endpoint version 3

Important

The task metadata version 3 endpoint is no longer being actively maintained. We recommend that you update the task metadata version 4 endpoint to get the latest metadata endpoint information. For more information, see the section called "Task metadata endpoint version 4".

If you are using Amazon ECS tasks hosted on AWS Fargate, see Amazon ECS task metadata endpoint version 3 for tasks on Fargate.

Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called `ECS_CONTAINER_METADATA_URI` into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and Docker stats are available to tasks. For tasks that use the `bridge` network mode, network metrics are available when querying the `/stats` endpoints.

The task metadata endpoint version 3 feature is enabled by default for tasks that use the Fargate launch type on platform version v1.3.0 or later and tasks that use the EC2 launch type and are launched on Amazon EC2 Linux infrastructure running at least version 1.21.0 of the Amazon ECS container agent or on Amazon EC2 Windows infrastructure running at least version `1.54.0` of the Amazon ECS container agent and use `awsvpc` network mode. For more information, see Amazon ECS Linux container instance management.

You can add support for this feature on older container instances by updating the agent to the latest version. For more information, see Updating the Amazon ECS container agent.

Important

For tasks using the Fargate launch type and platform versions prior to v1.3.0, the task metadata version 2 endpoint is supported. For more information, see Amazon ECS task metadata endpoint version 2.

Task Metadata endpoint version 3 paths

The following task metadata endpoints are available to containers:

`${ECS_CONTAINER_METADATA_URI}`

This path returns metadata JSON for the container.

`${ECS_CONTAINER_METADATA_URI}/task`

This path returns metadata JSON for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see Amazon ECS task metadata v3 JSON response.

`${ECS_CONTAINER_METADATA_URI}/taskWithTags`

This path returns the metadata for the task included in the `/task` endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API.

`${ECS_CONTAINER_METADATA_URI}/stats`

This path returns Docker stats JSON for the specific Docker container. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

`${ECS_CONTAINER_METADATA_URI}/task/stats`

This path returns Docker stats JSON for all of the containers associated with the task. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

Amazon ECS task metadata v3 JSON response

The following information is returned from the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI}/task`) JSON response.

`Cluster`

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

`TaskARN`

The Amazon Resource Name (ARN) of the task to which the container belongs.

`Family`

The family of the Amazon ECS task definition for the task.

`Revision`

The revision of the Amazon ECS task definition for the task.

`DesiredStatus`

The desired status for the task from Amazon ECS.

`KnownStatus`

The known status for the task from Amazon ECS.

`Limits`

The resource limits specified at the task level, such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

`PullStartedAt`

The timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

> **Note**
> The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 or later (Windows).

`Containers`

A list of container metadata for each container associated with the task.

`DockerId`

The Docker ID for the container.

`Name`

The name of the container as specified in the task definition.

`DockerName`

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

`Image`

The image for the container.

`ImageID`

The SHA-256 digest of the image manifest. This is the digest that can be used to pull the image using the format `repository-url/image@sha256:digest`.

`Ports`

Any ports exposed for the container. This parameter is omitted if there are no exposed ports.

`Labels`

Any labels applied to the container. This parameter is omitted if there are no labels applied.

`DesiredStatus`

The desired status for the container from Amazon ECS.

`KnownStatus`

The known status for the container from Amazon ECS.

`ExitCode`

The exit code for the container. This parameter is omitted if the container has not exited.

`Limits`

The resource limits specified at the container level, such as CPU (expressed in CPU units) and memory. This parameter is omitted if no resource limits are defined.

`CreatedAt`

The time stamp for when the container was created. This parameter is omitted if the container has not been created yet.

`StartedAt`

The time stamp for when the container started. This parameter is omitted if the container has not started yet.

`FinishedAt`

The time stamp for when the container stopped. This parameter is omitted if the container has not stopped yet.

`Type`

The type of the container. Containers that are specified in your task definition are of type `NORMAL`. You can ignore other container types, which are used for internal task resource provisioning by the Amazon ECS container agent.

`Networks`

The network information for the container, such as the network mode and IP address. This parameter is omitted if no network information is defined.

`ClockDrift`

The information about the difference between the reference time and the system time. This applies to the Linux operating system. This capability uses Amazon Time Sync Service to measure clock accuracy and provide the clock error bound for containers. For more information, see [Set the time for your Linux instance] in the *Amazon EC2 User Guide for Linux instances*.

`ReferenceTime`

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example `2021-09-07T16:57:44Z`.

`ClockErrorBound`

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

`ClockSynchronizationStatus`

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.

The valid values are `SYNCHRONIZED` and `NOT_SYNCHRONIZED`.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Amazon ECS task metadata v3 examples

The following examples show sample outputs from the task metadata endpoints.

Example Container Metadata Response

When querying the `${ECS_CONTAINER_METADATA_URI}` endpoint you are returned only metadata about the container itself. The following is an example output.


```
{
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
    "Name": "nginx-curl",
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
    "Image": "nrdlngr/nginx-curl",
    "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 512,
        "Memory": 512
    },
    "CreatedAt": "2018-02-01T20:55:10.554941919Z",
    "StartedAt": "2018-02-01T20:55:11.064236631Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.106"
            ]
        }
    ]
}
```


Example task metadata response

When querying the `${ECS_CONTAINER_METADATA_URI}/task` endpoint you are returned metadata about the task the container is part of. The following is an example output.

The following JSON response is for a single-container task.


```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
```

```
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS task metadata endpoint version 2

> **Important**
>
> The task metadata version 2 endpoint is no longer being actively maintained. We recommend that you update the task metadata version 4 endpoint to get the latest metadata endpoint information. For more information, see the section called "Task metadata endpoint version 4".

Beginning with version 1.17.0 of the Amazon ECS container agent, various task metadata and Docker stats are available to tasks that use the `awsvpc` network mode at an HTTP endpoint that is provided by the Amazon ECS container agent.

All containers belonging to tasks that are launched with the `awsvpc` network mode receive a local IPv4 address within a predefined link-local address range. When a container queries the metadata endpoint, the Amazon ECS container agent can determine which task the container belongs to based on its unique IP address, and metadata and stats for that task are returned.

Enabling task metadata

The task metadata version 2 feature is enabled by default for the following:

- Tasks using the Fargate launch type that use platform version v1.1.0 or later. For more information, see Fargate platform versions for Amazon ECS.
- Tasks using the EC2 launch type that also use the `awsvpc` network mode and are launched on Amazon EC2 Linux infrastructure running at least version 1.17.0 of the Amazon ECS container agent or on Amazon EC2 Windows infrastructure running at least version `1.54.0` of the Amazon ECS container agent. For more information, see Amazon ECS Linux container instance management.

You can add support for this feature on older container instances by updating the agent to the latest version. For more information, see Updating the Amazon ECS container agent.

Task metadata endpoint paths

The following API endpoints are available to containers:

`169.254.170.2/v2/metadata`

This endpoint returns metadata JSON for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see Task metadata JSON response.

`169.254.170.2/v2/metadata/<container-id>`

This endpoint returns metadata JSON for the specified Docker container ID.

`169.254.170.2/v2/metadata/taskWithTags`

This path returns the metadata for the task included in the `/task` endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API.

`169.254.170.2/v2/stats`

This endpoint returns Docker stats JSON for all of the containers associated with the task. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

`169.254.170.2/v2/stats/<container-id>`

This endpoint returns Docker stats JSON for the specified Docker container ID. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

Task metadata JSON response

The following information is returned from the task metadata endpoint (`169.254.170.2/v2/metadata`) JSON response.

`Cluster`

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

`TaskARN`

The Amazon Resource Name (ARN) of the task to which the container belongs.

`Family`

The family of the Amazon ECS task definition for the task.

`Revision`

The revision of the Amazon ECS task definition for the task.

`DesiredStatus`

The desired status for the task from Amazon ECS.

`KnownStatus`

The known status for the task from Amazon ECS.

`Limits`

The resource limits specified at the task level, such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

`PullStartedAt`

The timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

> **Note**
>
> The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 or later (Windows).

`Containers`

A list of container metadata for each container associated with the task.

`DockerId`

The Docker ID for the container.

`Name`

The name of the container as specified in the task definition.

`DockerName`

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

`Image`

The image for the container.

`ImageID`

The SHA-256 digest of the image manifest. This is the digest that can be used to pull the image using the format `repository-url/image@sha256:digest`.

`Ports`

Any ports exposed for the container. This parameter is omitted if there are no exposed ports.

`Labels`

Any labels applied to the container. This parameter is omitted if there are no labels applied.

`DesiredStatus`

The desired status for the container from Amazon ECS.

`KnownStatus`

The known status for the container from Amazon ECS.

`ExitCode`

The exit code for the container. This parameter is omitted if the container has not exited.

`Limits`

The resource limits specified at the container level, such as CPU (expressed in CPU units) and memory. This parameter is omitted if no resource limits are defined.

`CreatedAt`

The time stamp for when the container was created. This parameter is omitted if the container has not been created yet.

`StartedAt`

The time stamp for when the container started. This parameter is omitted if the container has not started yet.

`FinishedAt`

The time stamp for when the container stopped. This parameter is omitted if the container has not stopped yet.

`Type`

The type of the container. Containers that are specified in your task definition are of type `NORMAL`. You can ignore other container types, which are used for internal task resource provisioning by the Amazon ECS container agent.

`Networks`

The network information for the container, such as the network mode and IP address. This parameter is omitted if no network information is defined.

`ClockDrift`

The information about the difference between the reference time and the system time. This applies to the Linux operating system. This capability uses Amazon Time Sync Service to measure clock accuracy and provide the clock error bound for containers. For more information, see Set the time for your Linux instance in the *Amazon EC2 User Guide for Linux instances*.

`ReferenceTime`

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example `2021-09-07T16:57:44Z`.

`ClockErrorBound`

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

`ClockSynchronizationStatus`

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.

The valid values are `SYNCHRONIZED` and `NOT_SYNCHRONIZED`.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Example task metadata response

The following JSON response is for a single-container task.


```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
```

```
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS task metadata available for tasks on Fargate

Amazon ECS on Fargate provides a method to retrieve various metadata, network metrics, and Docker stats about your containers and the tasks they are a part of. This is referred to as the *task metadata endpoint*. The following task metadata endpoint versions are available for Amazon ECS on Fargate tasks:

- Task metadata endpoint version 4 – Available for tasks that use platform version 1.4.0 or later.
- Task metadata endpoint version 3 – Available for tasks that use platform version 1.1.0 or later.

All containers belonging to tasks that are launched with the `awsvpc` network mode receive a local IPv4 address within a predefined link-local address range. When a container queries the metadata endpoint, the container agent can determine which task the container belongs to based on its unique IP address, and metadata and stats for that task are returned.

Topics

Amazon ECS task metadata endpoint version 4 for tasks on Fargate

> **Important**
>
> If you are using Amazon ECS tasks hosted on Amazon EC2 instances, see Amazon ECS task metadata endpoint.

Beginning with Fargate platform version `1.4.0`, an environment variable named `ECS_CONTAINER_METADATA_URI_V4` is injected into each container in a task. When you query the task metadata endpoint version 4, various task metadata and Docker stats are available to tasks.

The task metadata endpoint version 4 functions like the version 3 endpoint but provides additional network metadata for your containers and tasks. Additional network metrics are available when querying the `/stats` endpoints as well.

The task metadata endpoint is on by default for all Amazon ECS tasks run on AWS Fargate that use platform version `1.4.0` or later.

> **Note**
>
> To avoid the need to create new task metadata endpoint versions in the future, additional metadata may be added to the version 4 output. We will not remove any existing metadata or change the metadata field names.

Fargate task metadata endpoint version 4 paths

The following task metadata endpoints are available to containers:

`${ECS_CONTAINER_METADATA_URI_V4}`

This path returns metadata for the container.

`${ECS_CONTAINER_METADATA_URI_V4}/task`

This path returns metadata for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see Amazon ECS task metadata v4 JSON response for tasks on Fargate.

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

This path returns Docker stats for the Docker container. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

> **Note**
>
> Amazon ECS tasks on AWS Fargate require that the container run for ~1 second prior to returning the container stats.

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

This path returns Docker stats for all of the containers associated with the task. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

> **Note**
>
> Amazon ECS tasks on AWS Fargate require that the container run for ~1 second prior to returning the container stats.

Amazon ECS task metadata v4 JSON response for tasks on Fargate

The following metadata is returned in the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON response.

`Cluster`

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

`VPCID`

The VPC ID of the Amazon EC2 container instance. This field only appears for Amazon EC2 instances.

> **Note**
>
> The `VPCID` metadata is only included when using Amazon ECS container agent version `1.63.1` or later.

`TaskARN`

The Amazon Resource Name (ARN) of the task to which the container belongs.

`Family`

The family of the Amazon ECS task definition for the task.

`Revision`

The revision of the Amazon ECS task definition for the task.

`DesiredStatus`

The desired status for the task from Amazon ECS.

`KnownStatus`

The known status for the task from Amazon ECS.

`Limits`

The resource limits specified at the task levels such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

`PullStartedAt`

The timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

> **Note**
>
> The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 (Windows).

`LaunchType`

The launch type the task is using. When using cluster capacity providers, this indicates whether the task is using Fargate or EC2 infrastructure.

> **Note**
>
> This `LaunchType` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later (Linux) or 1.0.0 or later (Windows).

`EphemeralStorageMetrics`

The reserved size and current usage of the ephemeral storage of this task.

> **Note**
>
> Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as `df`.

`Utilized`

The current ephemeral storage usage (in MiB) of this task.

`Reserved`

The reserved ephemeral storage (in MiB) of this task. The size of the ephemeral storage can't be changed in a running task. You can specify the `ephermalStorage` object in your task definition to change the amount of ephemeral storage. The `ephermalStorage` is specified in GiB, not MiB. The `ephermalStorage` and the `EphemeralStorageMetrics` are only available on Fargate Linux platform version 1.4.0 or later.

`Containers`

A list of container metadata for each container associated with the task.

`DockerId`

The Docker ID for the container.

When you use Fargate, the id is a 32-digit hex followed by a 10 digit number.

`Name`

The name of the container as specified in the task definition.

`DockerName`

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

`Image`

The image for the container.

`ImageID`

The SHA-256 digest of the image manifest. This is the digest that can be used to pull the image using the format `repository-url/image@sha256:digest`.

`Ports`

Any ports exposed for the container. This parameter is omitted if there are no exposed ports.

`Labels`

Any labels applied to the container. This parameter is omitted if there are no labels applied.

`DesiredStatus`

The desired status for the container from Amazon ECS.

`KnownStatus`

The known status for the container from Amazon ECS.

`ExitCode`

The exit code for the container. This parameter is omitted if the container has not exited.

`Limits`

The resource limits specified at the container level such as CPU (expressed in CPU units) and memory. This parameter is omitted if no resource limits are defined.

`CreatedAt`

The time stamp for when the container was created. This parameter is omitted if the container has not been created yet.

`StartedAt`

The time stamp for when the container started. This parameter is omitted if the container has not started yet.

`FinishedAt`

The time stamp for when the container stopped. This parameter is omitted if the container has not stopped yet.

`Type`

The type of the container. Containers that are specified in your task definition are of type `NORMAL`. You can ignore other container types, which are used for internal task resource provisioning by the Amazon ECS container agent.

`LogDriver`

The log driver the container is using.

> **Note**
>
> This `LogDriver` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later.

`LogOptions`

The log driver options defined for the container.

> **Note**
>
> This `LogOptions` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later.

`ContainerARN`

The Amazon Resource Name (ARN) of the container.

> **Note**
>
> This `ContainerARN` metadata is only included when using Amazon ECS Linux container agent version `1.45.0` or later.

`Networks`

The network information for the container, such as the network mode and IP address. This parameter is omitted if no network information is defined.

`Snapshotter`

The snapshotter that was used by containerd to download this container image. Valid values are `overlayfs`, which is the default, and `soci`, used when lazy loading with a SOCI index. This parameter is only available for tasks that run on Linux platform version `1.4.0`.

`RestartCount`

The number of times the container has been restarted.

> **Note**
>
> The `RestartCount` metadata is included only if a restart policy is enabled for the container. For more information, see Restart individual containers in Amazon ECS tasks with container restart policies.

`ClockDrift`

The information about the difference between the reference time and the system time. This capability uses Amazon Time Sync Service to measure clock accuracy and provide the clock error bound for containers. For more information, see Set the time for your Linux instance in the *Amazon EC2 User Guide for Linux instances*.

`ReferenceTime`

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example `2021-09-07T16:57:44Z`.

`ClockErrorBound`

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

`ClockSynchronizationStatus`

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.

The valid values are `SYNCHRONIZED` and `NOT_SYNCHRONIZED`.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Amazon ECS task metadata v4 examples for tasks on Fargate

The following examples show sample outputs from the task metadata endpoints for Amazon ECS tasks run on AWS Fargate.

From the container, you can use curl followed by the task meta data endpoint to query the endpoint for example `curl ${ECS_CONTAINER_METADATA_URI_V4}/task`.

Example container metadata response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}` endpoint you are returned only metadata about the container itself. The following is an example output.


```
{
    "DockerId": "cd189a933e5849daa93386466019ab50-2495160603",
    "Name": "curl",
    "DockerName": "curl",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID":
 "sha256:25f3695bedfb454a50f12d127839a68ad3caf91e451c1da073db34c542c4d2cb",
    "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/
default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/
cd189a933e5849daa93386466019ab50",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "2"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-08T20:09:11.44527186Z",
    "StartedAt": "2020-10-08T20:09:11.44527186Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "192.0.2.3"
            ],
```

```
            "AttachmentIndex": 0,
            "MACAddress": "0a:de:f6:10:51:e5",
            "IPv4SubnetCIDRBlock": "192.0.2.0/24",
            "DomainNameServers": [
                "192.0.2.2"
            ],
            "DomainNameSearchList": [
                "us-west-2.compute.internal"
            ],
            "PrivateDNSName": "ip-10-0-0-222.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "192.0.2.0/24"
        }
    ],
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/05966557-
f16c-49cb-9352-24b3a0dcd0e1",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/containerlogs",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/cd189a933e5849daa93386466019ab50"
    },
    "LogDriver": "awslogs",
    "Snapshotter": "overlayfs"
}
```

Amazon ECS task metadata v4 examples for tasks on Fargate

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/task` endpoint you are returned metadata about the task the container is part of. The following is an example output.

```
{
  "Cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/MyEmptyCluster",
  "TaskARN": "arn:aws:ecs:us-east-1:123456789012:task/MyEmptyCluster/
bfa2636268144d039771334145e490c5",
  "Family": "sample-fargate",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Limits": {
    "CPU": 0.25,
    "Memory": 512
  },
  "PullStartedAt": "2023-07-21T15:45:33.532811081Z",
```

```
  "PullStoppedAt": "2023-07-21T15:45:38.541068435Z",
  "AvailabilityZone": "us-east-1d",
  "Containers": [
    {
      "DockerId": "bfa2636268144d039771334145e490c5-1117626119",
      "Name": "curl-image",
      "DockerName": "curl-image",
      "Image": "curlimages/curl",
      "ImageID":
 "sha256:daf3f46a2639c1613b25e85c9ee4193af8a1d538f92483d67f9a3d7f21721827",
      "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/
MyEmptyCluster",
        "com.amazonaws.ecs.container-name": "curl-image",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-1:123456789012:task/
MyEmptyCluster/bfa2636268144d039771334145e490c5",
        "com.amazonaws.ecs.task-definition-family": "sample-fargate",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": { "CPU": 128 },
      "CreatedAt": "2023-07-21T15:45:44.91368314Z",
      "StartedAt": "2023-07-21T15:45:44.91368314Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": ["172.31.42.189"],
          "AttachmentIndex": 0,
          "MACAddress": "0e:98:9f:33:76:d3",
          "IPv4SubnetCIDRBlock": "172.31.32.0/20",
          "DomainNameServers": ["172.31.0.2"],
          "DomainNameSearchList": ["ec2.internal"],
          "PrivateDNSName": "ip-172-31-42-189.ec2.internal",
          "SubnetGatewayIpv4Address": "172.31.32.1/20"
        }
      ],
      "ContainerARN": "arn:aws:ecs:us-east-1:123456789012:container/MyEmptyCluster/
bfa2636268144d039771334145e490c5/da6cccf7-1178-400c-afdf-7536173ee209",
      "Snapshotter": "overlayfs"
    },
    {
      "DockerId": "bfa2636268144d039771334145e490c5-3681984407",
```

```
      "Name": "fargate-app",
      "DockerName": "fargate-app",
      "Image": "public.ecr.aws/docker/library/httpd:latest",
      "ImageID":
 "sha256:8059bdd0058510c03ae4c808de8c4fd2c1f3c1b6d9ea75487f1e5caa5ececa02",
      "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-east-1:123456789012:cluster/
MyEmptyCluster",
        "com.amazonaws.ecs.container-name": "fargate-app",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-1:123456789012:task/
MyEmptyCluster/bfa2636268144d039771334145e490c5",
        "com.amazonaws.ecs.task-definition-family": "sample-fargate",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": { "CPU": 2 },
      "CreatedAt": "2023-07-21T15:45:44.954460255Z",
      "StartedAt": "2023-07-21T15:45:44.954460255Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": ["172.31.42.189"],
          "AttachmentIndex": 0,
          "MACAddress": "0e:98:9f:33:76:d3",
          "IPv4SubnetCIDRBlock": "172.31.32.0/20",
          "DomainNameServers": ["172.31.0.2"],
          "DomainNameSearchList": ["ec2.internal"],
          "PrivateDNSName": "ip-172-31-42-189.ec2.internal",
          "SubnetGatewayIpv4Address": "172.31.32.1/20"
        }
      ],
      "ContainerARN": "arn:aws:ecs:us-east-1:123456789012:container/MyEmptyCluster/
bfa2636268144d039771334145e490c5/f65b461d-aa09-4acb-a579-9785c0530cbc",
      "Snapshotter": "overlayfs"
    }
  ],
  "LaunchType": "FARGATE",
  "ClockDrift": {
    "ClockErrorBound": 0.446931,
    "ReferenceTimestamp": "2023-07-21T16:09:17Z",
    "ClockSynchronizationStatus": "SYNCHRONIZED"
  },
```

```
  "EphemeralStorageMetrics": {
    "Utilized": 261,
    "Reserved": 20496
  }
}
```

Example task stats response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/task/stats` endpoint you are returned network metrics about the task the container is part of. The following is an example output.


```
{
  "3d1f891cded94dc795608466cce8ddcf-464223573": {
    "read": "2020-10-08T21:24:44.938937019Z",
    "preread": "2020-10-08T21:24:34.938633969Z",
    "pids_stats": {},
    "blkio_stats": {
      "io_service_bytes_recursive": [
        {
          "major": 202,
          "minor": 26368,
          "op": "Read",
          "value": 638976
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Write",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Sync",
          "value": 638976
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Async",
          "value": 0
        },
```

```
          {
            "major": 202,
            "minor": 26368,
            "op": "Total",
            "value": 638976
          }
        ],
        "io_serviced_recursive": [
          {
            "major": 202,
            "minor": 26368,
            "op": "Read",
            "value": 12
          },
          {
            "major": 202,
            "minor": 26368,
            "op": "Write",
            "value": 0
          },
          {
            "major": 202,
            "minor": 26368,
            "op": "Sync",
            "value": 12
          },
          {
            "major": 202,
            "minor": 26368,
            "op": "Async",
            "value": 0
          },
          {
            "major": 202,
            "minor": 26368,
            "op": "Total",
            "value": 12
          }
        ],
        "io_queue_recursive": [],
        "io_service_time_recursive": [],
        "io_wait_time_recursive": [],
        "io_merged_recursive": [],
        "io_time_recursive": [],
```

```
        "sectors_recursive": []
      },
      "num_procs": 0,
      "storage_stats": {},
      "cpu_stats": {
        "cpu_usage": {
          "total_usage": 1137691504,
          "percpu_usage": [
            696479228,
            441212276,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0
          ],
          "usage_in_kernelmode": 80000000,
          "usage_in_usermode": 810000000
        },
        "system_cpu_usage": 9393210000000,
        "online_cpus": 2,
        "throttling_data": {
          "periods": 0,
          "throttled_periods": 0,
          "throttled_time": 0
        }
      },
      "precpu_stats": {
        "cpu_usage": {
          "total_usage": 1136624601,
          "percpu_usage": [
            695639662,
            440984939,
            0,
            0,
            0,
```

```
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0
        ],
        "usage_in_kernelmode": 80000000,
        "usage_in_usermode": 810000000
      },
      "system_cpu_usage": 9373330000000,
      "online_cpus": 2,
      "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
      }
    },
    "memory_stats": {
      "usage": 6504448,
      "max_usage": 8458240,
      "stats": {
        "active_anon": 1675264,
        "active_file": 557056,
        "cache": 651264,
        "dirty": 0,
        "hierarchical_memory_limit": 536870912,
        "hierarchical_memsw_limit": 9223372036854772000,
        "inactive_anon": 0,
        "inactive_file": 3088384,
        "mapped_file": 430080,
        "pgfault": 11034,
        "pgmajfault": 5,
        "pgpgin": 8436,
        "pgpgout": 7137,
        "rss": 4669440,
        "rss_huge": 0,
        "total_active_anon": 1675264,
        "total_active_file": 557056,
        "total_cache": 651264,
```

```
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 3088384,
        "total_mapped_file": 430080,
        "total_pgfault": 11034,
        "total_pgmajfault": 5,
        "total_pgpgin": 8436,
        "total_pgpgout": 7137,
        "total_rss": 4669440,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
      },
      "limit": 9223372036854772000
    },
    "name": "curltest",
    "id": "3d1f891cded94dc795608466cce8ddcf-464223573",
    "networks": {
      "eth1": {
        "rx_bytes": 2398415937,
        "rx_packets": 1898631,
        "rx_errors": 0,
        "rx_dropped": 0,
        "tx_bytes": 1259037719,
        "tx_packets": 428002,
        "tx_errors": 0,
        "tx_dropped": 0
      }
    },
    "network_rate_stats": {
      "rx_bytes_per_sec": 43.298687872232854,
      "tx_bytes_per_sec": 215.39347269466413
    }
  }
}
```

Amazon ECS task metadata endpoint version 3 for tasks on Fargate

> **Important**
>
> The task metadata version 3 endpoint is no longer being actively maintained. We recommend that you update the task metadata version 4 endpoint to get the latest metadata endpoint information. For more information, see the section called "Task metadata endpoint version 4 for tasks on Fargate".

Beginning with Fargate platform version `1.1.0`, an environment variable named `ECS_CONTAINER_METADATA_URI` is injected into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and Docker stats are available to tasks.

The task metadata endpoint feature is enabled by default for Amazon ECS tasks hosted on Fargate that use platform version `1.1.0` or later. For more information, see Fargate platform versions for Amazon ECS.

Task metadata endpoint paths for tasks on Fargate

The following API endpoints are available to containers:

`${ECS_CONTAINER_METADATA_URI}`

This path returns metadata JSON for the container.

`${ECS_CONTAINER_METADATA_URI}/task`

This path returns metadata JSON for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see Amazon ECS task metadata v3 JSON response for tasks on Fargate.

`${ECS_CONTAINER_METADATA_URI}/stats`

This path returns Docker stats JSON for the specific Docker container. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

`${ECS_CONTAINER_METADATA_URI}/task/stats`

This path returns Docker stats JSON for all of the containers associated with the task. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

Amazon ECS task metadata v3 JSON response for tasks on Fargate

The following information is returned from the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI}/task`) JSON response.

`Cluster`

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

`TaskARN`

The Amazon Resource Name (ARN) of the task to which the container belongs.

`Family`

The family of the Amazon ECS task definition for the task.

`Revision`

The revision of the Amazon ECS task definition for the task.

`DesiredStatus`

The desired status for the task from Amazon ECS.

`KnownStatus`

The known status for the task from Amazon ECS.

`Limits`

The resource limits specified at the task level, such as CPU (expressed in vCPUs) and memory. This parameter is omitted if no resource limits are defined.

`PullStartedAt`

The timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

> **Note**
>
> The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later (Linux) or 1.0.0 or later (Windows).

`Containers`

A list of container metadata for each container associated with the task.

`DockerId`

The Docker ID for the container.

`Name`

The name of the container as specified in the task definition.

`DockerName`

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

`Image`

The image for the container.

`ImageID`

The SHA-256 digest of the image manifest. This is the digest that can be used to pull the image using the format `repository-url/image@sha256:digest`.

`Ports`

Any ports exposed for the container. This parameter is omitted if there are no exposed ports.

`Labels`

Any labels applied to the container. This parameter is omitted if there are no labels applied.

`DesiredStatus`

The desired status for the container from Amazon ECS.

`KnownStatus`

The known status for the container from Amazon ECS.

`ExitCode`

The exit code for the container. This parameter is omitted if the container has not exited.

`Limits`

The resource limits specified at the container level, such as CPU (expressed in CPU units) and memory. This parameter is omitted if no resource limits are defined.

`CreatedAt`

The time stamp for when the container was created. This parameter is omitted if the container has not been created yet.

`StartedAt`

The time stamp for when the container started. This parameter is omitted if the container has not started yet.

`FinishedAt`

The time stamp for when the container stopped. This parameter is omitted if the container has not stopped yet.

`Type`

The type of the container. Containers that are specified in your task definition are of type `NORMAL`. You can ignore other container types, which are used for internal task resource provisioning by the Amazon ECS container agent.

`Networks`

The network information for the container, such as the network mode and IP address. This parameter is omitted if no network information is defined.

`ClockDrift`

The information about the difference between the reference time and the system time. This applies to the Linux operating system. This capability uses Amazon Time Sync Service to measure clock accuracy and provide the clock error bound for containers. For more information, see Set the time for your Linux instance in the *Amazon EC2 User Guide for Linux instances*.

`ReferenceTime`

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example `2021-09-07T16:57:44Z`.

`ClockErrorBound`

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

`ClockSynchronizationStatus`

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.

The valid values are `SYNCHRONIZED` and `NOT_SYNCHRONIZED`.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Amazon ECS task metadata v3 examples for tasks on Fargate

The following JSON response is for a single-container task.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
```

```
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID":
 "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
```

```
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS container introspection

The Amazon ECS container agent provides an API operation for gathering details about the container instance on which the agent is running and the associated tasks running on that instance. You can use the **curl** command from within the container instance to query the Amazon ECS container agent (port 51678) and return container instance metadata or task information.

> **Important**
>
> Your container instance must have an IAM role that allows access to Amazon ECS in order to retrieve the metadata. For more information, see Amazon ECS container instance IAM role.

To view container instance metadata, log in to your container instance via SSH and run the following command. Metadata includes the container instance ID, the Amazon ECS cluster in which the container instance is registered, and the Amazon ECS container agent version information.

```
curl -s http://localhost:51678/v1/metadata | python3 -mjson.tool
```

Output:

```
{
    "Cluster": "cluster_name",
    "ContainerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-instance/cluster_name/container_instance_id",
    "Version": "Amazon ECS Agent - v1.30.0 (02ff320c)"
```

```
}
```

To view information about all of the tasks that are running on a container instance, log in to your container instance via SSH and run the following command:

```
curl http://localhost:51678/v1/tasks
```

Output:


```
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/example5-58ff-46c9-
ae05-543f8example",
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Family": "hello_world",
            "Version": "8",
            "Containers": [
                {
          "DockerId":
 "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
          "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
          "Name": "mysql",
          "CreatedAt": "2023-10-08T20:09:11.44527186Z",
          "StartedAt": "2023-10-08T20:09:11.44527186Z",
          "ImageID":
 "sha256:2ae34abc2ed0a22e280d17e13f9c01aaf725688b09b7a1525d1a2750e2c0d1de"
        },
        {
          "DockerId":
 "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
          "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",
          "Name": "wordpress"
        }
    ]
}
    ]
}
```

You can view information for a particular task that is running on a container instance. To specify a specific task or container, append one of the following to the request:

- The task ARN (`?taskarn=task_arn`)
- The Docker ID for a container (`?dockerid=docker_id`)

To get task information with a container's Docker ID, log in to your container instance via SSH and run the following command.

Note

Amazon ECS container agents before version 1.14.2 require full Docker container IDs for the introspection API, not the short version that is shown with **docker ps**. You can get the full Docker ID for a container by running the **docker ps --no-trunc** command on the container instance.

```
curl http://localhost:51678/v1/tasks?dockerid=79c796ed2a7f
```

Output:

```
{
    "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/e01d58a8-151b-40e8-bc01-22647b9ecfec",
    "Containers": [
        {
            "DockerId": "79c796ed2a7f864f485c76f83f3165488097279d296a7c05bd5201a1c69b2920",
            "DockerName": "ecs-nginx-efs-2-nginx-9ac0808dd0afa495f001",
            "Name": "nginx",
            "CreatedAt": "2023-10-08T20:09:11.44527186Z",
            "StartedAt": "2023-10-08T20:09:11.44527186Z",
            "ImageID": "sha256:2ae34abc2ed0a22e280d17e13f9c01aaf725688b09b7a1525d1a2750e2c0d1de"
        }
    ],
    "DesiredStatus": "RUNNING",
    "Family": "nginx-efs",
    "KnownStatus": "RUNNING",
```

```
    "Version": "2"
}
```

Identify unauthorized behavior using Runtime Monitoring

Amazon GuardDuty is a threat detection service that helps protect your accounts, containers, workloads, and the data within your AWS environment. Using machine learning (ML) models, and anomaly and threat detection capabilities, GuardDuty continuously monitors different log sources and runtime activity to identify and prioritize potential security risks and malicious activities in your environment.

Runtime Monitoring in GuardDuty protects workloads running on Fargate and EC2 container instances by continuously monitoring AWS log and networking activity to identify malicious or unauthorized behavior. Runtime Monitoring uses a lightweight, fully managed GuardDuty security agent that analyzes on-host behavior, such as file access, process execution, and network connections. This covers issues including escalation of privileges, use of exposed credentials, or communication with malicious IP addresses, domains, and the presence of malware on your Amazon EC2 instances and container workloads. For more information, see GuardDuty Runtime Monitoring in the *GuardDuty User Guide*.

Your security administrator enables Runtime Monitoring for a single or multiple accounts in AWS Organizations for GuardDuty. They also select whether GuardDuty automatically deploys the GuardDuty security agent when you use Fargate. All your clusters are automatically protected, and GuardDuty manages the security agent on your behalf.

You can also manually configure the GuardDuty security agent in the following cases:

- You use EC2 container instances
- You need granular control to enable Runtime Monitoring at the cluster level

To use Runtime Monitoring, you must configure the clusters that are protected, and install and manage the GuardDuty security agent on your EC2 container instances.

How Runtime Monitoring works with Amazon ECS

Runtime Monitoring uses a lightweight GuardDuty security agent that monitors Amazon ECS workload activity for how applications are requesting, gaining access and consuming underlying system resources.

For Fargate tasks, the GuardDuty security agent runs as a sidecar container for each task.

For EC2 container instances, the GuardDuty security agent runs as a process on the instance.

The GuardDuty security agent collects data from the following resources, and then sends the data to GuardDuty to process. You can view the findings in the GuardDuty console. You can also send them to other AWS services such as AWS Security Hub, or a third-party security vendor for aggregation and remediation. For information about how to view and manage findings, see Managing Amazon GuardDuty findings in the *Amazon GuardDuty User Guide*.

- Responses from the following Amazon ECS API calls:
  - DescribeClusters

    The response parameters include the Runtime Monitoring tag (when the tag is set) when you use the `--include TAGS` option.
  - DescribeTasks

    For the Fargate launch type, the response parameters include the GuardDuty sidecar container.
  - ListAccountSettings

    The response parameters include the Runtime Monitoring account setting, which is set by your security administrator.
- The container agent introspection data. For more information, see Amazon ECS container introspection.
- The task metadata endpoint for the launch type:
  - Amazon ECS task metadata endpoint version 4
  - Amazon ECS task metadata endpoint version 4 for tasks on Fargate

Considerations

Consider the following when using Runtime Monitoring:

- Runtime Monitoring has a cost associated with it. For more information, see Amazon GuardDuty Pricing.
- Runtime Monitoring is not supported on Amazon ECS Anywhere.
- Runtime Monitoring is not supported for the Windows operating system.

- When you use Amazon ECS Exec on Fargate, you must specify the container name because the GuardDuty security agent runs as a sidecar container.
- You cannot use Amazon ECS Exec on the GuardDuty security agent sidecar container.
- The IAM user that controls Runtime Monitoring at the cluster level, must have the appropriate IAM permissions for tagging. For more information, see [IAM tutorial: Define permissions to access AWS resources based on tags]() in the *IAM User Guide*.
- Fargate tasks must use a task execution role. This role grants the tasks permission to retrieve, update, and manage the GuardDuty security agent, which is stored in an Amazon ECR private repository, on your behalf.

Resource utilization

The tag that you add to the cluster counts toward the cluster tag quota.

The GuardDuty agent sidecar container does not count toward the containers per task definition quota.

As with most security software, there is a slight overhead for GuardDuty. For information about the Fargate memory limits, see CPU and memory limits in the *GuardDuty User Guide*. For information about the Amazon EC2 memory limits, see CPU and memory limit for GuardDuty agent.

Runtime Monitoring for Amazon ECS Fargate workloads

If you use EC2 container instances, you must manually configure Runtime Monitoring. For more information, see Runtime Monitoring for EC2 workloads on Amazon ECS.

You can have GuardDuty manage the security agent on your container instances. This option is only available for Fargate. This option (GuardDuty agent management) is available in GuardDuty

When you use GuardDuty agent management, GuardDuty performs the following operations:

- Creates VPC endpoints for GuardDuty for each VPC that hosts a cluster.
- Retrieves, and installs the latest GuardDuty security agent as a sidecar container on all new standalone Fargate tasks, and new service deployments.

  A new service deployment happens the first time you launch a service, or when you update an existing service with the **force new deployment** option.

Turning on Runtime Monitoring for Amazon ECS

You can configure GuardDuty to automatically manage the security agen for all your Fargate clusters.

Prerequisites

The following are prerequisites for using Runtime Monitoring:

- The Fargate platform version must be `1.4.0` or later for Linux.
- IAM roles and permissions for Amazon ECS:
  - Fargate tasks must use a task execution role. This role grants the tasks permission to retrieve, update, and manage the GuardDuty security agent on your behalf. For more information see Amazon ECS task execution IAM role.
  - You control Runtime Monitoring for a cluster with a pre-defined tag. If your access policies restrict access based on tags, you must grant explicit permissions to your IAM users to tag clusters. For more information, see IAM tutorial: Define permissions to access AWS resources based on tags in the *IAM User Guide*.
- Connecting to the Amazon ECR repository:

  The GuardDuty security agent is stored in an Amazon ECR repository. Each standalone and service task must have access to the repository. You can use one of the following options:
  - For tasks in public subnets, you can either use a public IP address for the task, or create a VPC endpoint for Amazon ECR in the subnet where the task runs. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.
  - For tasks in private subnets, you can use a Network Address Translation (NAT) gateway, or create a VPC endpoint for Amazon ECR in the subnet where the task runs.

    For more information, see Private subnet and NAT gateway.
- You must have the `AWSServiceRoleForAmazonGuardDuty` role for GuardDuty. For more information, see Service-linked role permissions for GuardDuty in the *Amazon GuardDuty User Guide*.
- Any files that you want to protect with Runtime Monitoring must be accessible by the root user. If you manually changed the permissions of a file, you must set it to `755`.

The following are prerequisites for using Runtime Monitoring on EC2 container instances:

- You must use version `20230929` or later of the Amazon ECS-AMI.
- You must run Amazon ECS agent to version `1.77` or later on the container instances.
- You must use kernel version `5.10` or later.
- For information about the supported Linux operating systems and architectures, see Which operating models and workloads does GuardDuty Runtime Monitoring support.
- You can use Systems Manager to manage your container instances. For more information, see Setting up Systems Manager for EC2 instances in the *AWS Systems Manager Session Manager User Guide*.

Procedure

You enable Runtime Monitoring in GuardDuty. For information about how to enable the feature, see Enabling Runtime Monitoring in the *Amazon GuardDuty User Guide*.

Adding Runtime Monitoring to existing Amazon ECS Fargate tasks

When you turn on Runtime Monitoring, all new standalone tasks, and new service deployments in the cluster are protected automatically. In order to preserve the immutability constraint, existing tasks are not affected.

Prerequisites

1. Turn on Runtime Monitoring. For more information, see Turning on Runtime Monitoring for Amazon ECS.
2. Fargate tasks must use a task execution role. This role grants the tasks permission to retrieve, update, and manage the GuardDuty security agent on your behalf. For more information see Amazon ECS task execution IAM role.

Procedure

- To immediately protect a task, you need to perform one of the following actions:
  - For standalone tasks, stop the tasks, and then start them. For more information, see Stopping an Amazon ECS task and Running an application as an Amazon ECS task
  - For tasks that are part of a service, update the service with the "force new deployment" option. For more information, see Updating an Amazon ECS service.

Removing Runtime Monitoring from an Amazon ECS cluster

You might want to exclude certain clusters from protection, for example clusters that you use for testing. This causes GuardDuty to perform the following operations on resources in the cluster:

- No longer deploy the GuardDuty security agent to new standalone Fargate tasks, or new service deployments.

  In order to preserve the immutability constraint, existing tasks and deployments with Runtime Monitoring enabled are not affected.
- Stop billing and no longer accepts run time events for tasks.

Procedure

Perform the following steps to remove Runtime Monitoring from a cluster.

1. Use the Amazon ECS console or AWS CLI to set the `GuardDutyManaged` tag key on the cluster to `false`. For more information, see Updating a cluster or Working with tags using the CLI or API. Use the following values for the tag.

   > **Note**
   >
   > The Key and Value are case sensitive and must exactly match the strings.

   Key = `GuardDutyManaged`, Value = `false`
2. Delete the GuardDuty VPC endpoint for the cluster. For more information about how to delete VPC endpoints, see Delete an interface endpoint in the *AWS PrivateLink User Guide*.

Removing Runtime Monitoring for Amazon ECS from an account

When you no longer want to use Runtime Monitoring, disable the feature in GuardDuty. For information about how to disable the feature, see Enabling Runtime Monitoring in the *Amazon GuardDuty User Guide*.

GuardDuty performs the following operations:

- Deletes the VPC endpoints for GuardDuty for each VPC that hosts a cluster.

- No longer deploys the GuardDuty security agent to new standalone Fargate tasks, or new service deployments.

  In order to preserve the immutability constraint, existing tasks and deployments are not affected until they are stopped, replicated, or scaled.
- Stops billing and no longer accepts run time events for tasks.

Runtime Monitoring for EC2 workloads on Amazon ECS

Use this option when you use EC2 instances for your capacity, or when you need granular control of Runtime Monitoring at the cluster-level on Fargate.

You provision the clusters for Runtime Monitoring by adding a pre-defined tag.

For EC2 container instances, you download, install, and manage the GuardDuty security agent.

For Fargate, GuardDuty manages the security agent on your behalf.

Turning on Runtime Monitoring for Amazon ECS

You can turn on Runtime Monitoring for clusters with EC2 instances, or when you need granular control of Runtime Monitoring at the cluster-level on Fargate.

The following are prerequisites for using Runtime Monitoring:

- The Fargate platform version must be `1.4.0` or later for Linux.
- IAM roles and permissions for Amazon ECS:
  - Fargate tasks must use a task execution role. This role grants the tasks permission to retrieve, update, and manage the GuardDuty security agent on your behalf. For more information see [Amazon ECS task execution IAM role](#).
  - You control Runtime Monitoring for a cluster with a pre-defined tag. If your access policies restrict access based on tags, you must grant explicit permissions to your IAM users to tag clusters. For more information, see [IAM tutorial: Define permissions to access AWS resources based on tags](#) in the *IAM User Guide*.
- Connecting to the Amazon ECR repository:

  The GuardDuty security agent is stored in an Amazon ECR repository. Each standalone and service task must have access to the repository. You can use one of the following options:

  - For tasks in public subnets, you can either use a public IP address for the task, or create a VPC endpoint for Amazon ECR in the subnet where the task runs. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.
  - For tasks in private subnets, you can use a Network Address Translation (NAT) gateway, or create a VPC endpoint for Amazon ECR in the subnet where the task runs.

    For more information, see Private subnet and NAT gateway.
- You must have the `AWSServiceRoleForAmazonGuardDuty` role for GuardDuty. For more information, see Service-linked role permissions for GuardDuty in the *Amazon GuardDuty User Guide*.
- Any files that you want to protect with Runtime Monitoring must be accessible by the root user. If you manually changed the permissions of a file, you must set it to `755`.

The following are prerequisites for using Runtime Monitoring on EC2 container instances:

- You must use version `20230929` or later of the Amazon ECS-AMI.
- You must run Amazon ECS agent to version `1.77` or later on the container instances.
- You must use kernel version `5.10` or later.
- For information about the supported Linux operating systems and architectures, see Which operating models and workloads does GuardDuty Runtime Monitoring support.
- You can use Systems Manager to manage your container instances. For more information, see Setting up Systems Manager for EC2 instances in the *AWS Systems Manager Session Manager User Guide*.

You turn on Runtime Monitoring in GuardDuty. For information about how to enable the feature, see Enabling Runtime Monitoring in the *Amazon GuardDuty User Guide*.

Adding Runtime Monitoring an Amazon ECS cluster

Configure Runtime Monitoring for the cluster, and then install the GuardDuty security agent on your EC2 container instances.

Prerequisites

1. Turn on Runtime Monitoring. For more information, see Turning on Runtime Monitoring for Amazon ECS.
2. You control Runtime Monitoring for a cluster with a pre-defined tag. If your access policies restrict access based on tags, you must grant explicit permissions to your IAM users to tag clusters. For more information, see IAM tutorial: Define permissions to access AWS resources based on tags in the *IAM User Guide*.

Procedure

Perform the following operations to add Runtime Monitoring to a cluster.

1. Create a VPC endpoint for GuardDuty for each cluster VPC. For more information, see Creating Amazon VPC endpoint manually in the *GuardDuty User Guide*.
2. Configure the EC2 container instances.
   a. Update the Amazon ECS agent to version `1.77` or later on the EC2 container instances in the cluster. For more information see Updating the Amazon ECS container agent.
   b. Install the GuardDuty security agent on the EC2 container instances in the cluster. For more information, see Managing the security agent on an Amazon EC2 instance manually in the *GuardDuty User Guide*.

      All new and existing tasks, and deployments are immediately protected because the GuardDuty security agent runs as a process on the EC2 container instance.
3. Use the Amazon ECS console or AWS CLI to set the `GuardDutyManaged` tag key on the cluster to `true`. For more information, see Updating a cluster or Working with tags using the CLI or API. Use the following values for the tag.

   > **Note**
   > The Key and Value are case sensitive and must exactly match the strings.

   Key = `GuardDutyManaged`, Value = `true`

Adding Runtime Monitoring to existing Amazon ECS tasks

When you turn on Runtime Monitoring, all new standalone tasks, and new service deployments in the cluster are protected automatically. In order to preserve the immutability constraint, existing tasks are not affected.

Prerequisites

- Turn on Runtime Monitoring. For more information, see Turning on Runtime Monitoring for Amazon ECS.

Procedure

- To immediately protect a task, you need to perform one of the following actions:
  - For standalone tasks, stop the tasks, and then start them. For more information, see Stopping an Amazon ECS task and Running an application as an Amazon ECS task.
  - For tasks that are part of a service, update the service with the "force new deployment" option. For more information, see Updating an Amazon ECS service.

Removing Runtime Monitoring from an Amazon ECS cluster

You can remove Runtime Monitoring from a cluster. This causes GuardDuty to stop monitoring all resources in the cluster.

To remove Runtime Monitoring from a cluster

1. Use the Amazon ECS console or AWS CLI to set the `GuardDutyManaged` tag key on the cluster to `false`. For more information, see Updating a cluster or Working with tags using the CLI or API.

   > **Note**
   >
   > The Key and Value are case sensitive and must exactly match the strings.

   Key = `GuardDutyManaged`, Value = `false`
2. Uninstall the GuardDuty security agent on you EC2 container instances in the cluster.

For more information, see Uninstalling the security agent manually in the *GuardDuty User Guide*.

3. Delete the GuardDuty VPC endpoint for each cluster VPC. For more information about how to delete VPC endpoints, see Delete an interface endpoint in the *AWS PrivateLink User Guide*.

Updating the GuardDuty security agent on your Amazon ECS container instances

For information about how to update the GuardDuty security agent on your EC2 container instances, see Updating GuardDuty security agent in the *Amazon GuardDuty User Guide*.

Removing Runtime Monitoring for Amazon ECS from an account

When you no longer want to use Runtime Monitoring, disable the feature in GuardDuty. For information about how to disable the feature, see Enabling Runtime Monitoring in the *Amazon GuardDuty User Guide*.

Remove Runtime Monitoring from all clusters. For more information, see Removing Runtime Monitoring from an Amazon ECS cluster.

Runtime Monitoring Troubleshooting

You might need to troubleshoot or verify that Runtime Monitoring is enabled and running on your tasks and containers.

Topics

How can I tell if Runtime Monitoring is active on my account?

In the Amazon ECS console, the information is in on the **Account Settings** page.

You can also run `list-account-settings` with the `effective-settings` option.

```
aws ecs list-account-settings --effective-settings
```

Output

The setting with **name** set to `guardDutyActivate` and **value** set to `on` indicates that the account is configured. You must check with your GuardDuty administrator to see if the management is automatic or manual.

```
{
    "setting": {
        "name": "guardDutyActivate",
        "value": "enabled",
        "principalArn": "arn:aws:iam::123456789012:root",
        "type": "aws-managed"
    }
}
```

How can I tell if Runtime Monitoring is active on a cluster?

You can review the coverage statistics in the GuardDuty console. This includes information for the Amazon ECS resources associated with your own account or your member accounts is the percentage of the healthy clusters over all the clusters in the selected AWS Region. This includes the coverage for clusters that use the Fargate and EC2 launch types. For more information, see Reviewing coverage statistics in the *Amazon GuardDuty User Guide*.

How can I tell if the GuardDuty security agent is running on a Fargate task?

The GuardDuty security agent runs as a sidecar container for Fargate tasks.

In the Amazon ECS console, the sidecar is displayed under **Containers** on the **Task details** page.

You can run `describe-tasks` and look for the container with a **name** set to `aws-gd-agent` and the **lastStatus** set to `RUNNING`.

The following example shows the output for the default cluster for task `aws:ecs:us-east-1:123456789012:task/0b69d5c0-d655-4695-98cd-5d2d5EXAMPLE`.

```
aws ecs describe-tasks --cluster default --tasks aws:ecs:us-
east-1:123456789012:task/0b69d5c0-d655-4695-98cd-5d2d5EXAMPLE
```

Output

The container named `gd-agent` is in the `RUNNING` state.

```
"containers": [
      {
        "containerArn": "arn:aws:ecs:us-east-1:123456789012:container/4df26bb4-
f057-467b-a079-96167EXAMPLE",
        "taskArn": "arn:aws:ecs:us-east-1:123456789012:task/0b69d5c0-
d655-4695-98cd-5d2d5EXAMPLE",
        "lastStatus": "RUNNING",
        "healthStatus": "UNKNOWN",
        "memory": "string",
        "name": "aws-gd-agent"
      }
    ]
```

How can I tell if the GuardDuty security agent is running on an EC2 container instance?

Run the following command to view the status:

```
sudo systemctl status amazon-guardduty-agent
```

The log file is in the following location:

```
/var/log/amzn-guardduty-agent
```

What happens when there is no task execution role for a task running on the cluster?

For Fargate tasks, the task starts without the GuardDuty security agent sidecar container. The GuardDuty dashboard will show that the task is missing protection in the coverage statistics dashboard.

How can I tell if I have the correct permissions to tag clusters for Runtime Monitoring?

In order to tag a cluster, you must have the `ecs:TagResource` action for both `CreateCluster` and `UpdateCluster`.

The following is a snippet of an example policy.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "ecs:TagResource"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
             "ecs:CreateAction" : "CreateCluster",
             "ecs:CreateAction" : "UpdateCluster",
          }
       }
    }
  ]
}
```

What happens when there is no connection Amazon ECR?

For Fargate tasks, the task starts without the GuardDuty security agent sidecar container. The GuardDuty dashboard will show that the task is missing protection in the coverage statistics dashboard.

How do I address out of memory errors on my Fargate tasks after enabling Runtime Monitoring?

The GuardDuty security agent is a lightweight process. However, the process still consumes resources according to the size of the workload. We recommend using container resource tracking tooling, such as Amazon CloudWatch Container Insights to stage GuardDuty deployments in your cluster. These tools help you to discover the consumption profile of the GuardDuty security agent for your applications. You can then adjust your Fargate task size, if required, to avoid potential out of memory conditions.

Monitor Amazon ECS containers with ECS Exec

With Amazon ECS Exec, you can directly interact with containers without needing to first interact with the host container operating system, open inbound ports, or manage SSH keys. You can use ECS Exec to run commands in or get a shell to a container running on an Amazon EC2 instance or on AWS Fargate. This makes it easier to collect diagnostic information and quickly troubleshoot errors. For example, in a development context, you can use ECS Exec to easily interact with various process in your containers and troubleshoot your applications. And in production scenarios, you can use it to gain break-glass access to your containers to debug issues.

You can run commands in a running Linux or Windows container using ECS Exec from the Amazon ECS API, AWS Command Line Interface (AWS CLI), AWS SDKs, or the AWS Copilot CLI. For details on using ECS Exec, as well as a video walkthrough, using the AWS Copilot CLI, see the Copilot GitHub documentation.

You can also use ECS Exec to maintain stricter access control policies. By selectively turning on this feature, you can control who can run commands and on which tasks they can run those commands. With a log of each command and their output, you can use ECS Exec to view which tasks were run and you can use CloudTrail to audit who accessed a container.

Considerations

Consider the following when using ECS Exec:

- ECS Exec might not work as expected when running on operating systems not supported by Systems Manager. For information about the supported operating systems, see Operating system types in the *AWS Systems Manager User Guide*.
- ECS Exec is supported for tasks that run on the following infrastructure:
  - Linux containers on Amazon EC2 on any Amazon ECS-optimized AMI, including Bottlerocket
  - Linux and Windows containers on external instances (Amazon ECS Anywhere)
  - Linux and Windows containers on AWS Fargate
  - Windows containers on Amazon EC2 on the following Windows Amazon ECS-optimized AMIs (with the container agent version `1.56` or later):
    - Amazon ECS-optimized Windows Server 2022 Full AMI
    - Amazon ECS-optimized Windows Server 2022 Core AMI
    - Amazon ECS-optimized Windows Server 2019 Full AMI

  - Amazon ECS-optimized Windows Server 2019 Core AMI
  - Amazon ECS-optimized Windows Server 20H2 Core AMI
- If you configured an HTTP proxy for your task, set the `NO_PROXY` environment variable to `"NO_PROXY=169.254.169.254,169.254.170.2"` in order to bypass the proxy for EC2 instance metadata and IAM role traffic. If you don't configure the `NO_PROXY` environment variable, there can be failures when retrieving instance metadata or IAM role credentials from the metadata endpoint within the container. Setting the `NO_PROXY` environment variable as recommended filters the metadata and IAM traffic so that requests to `169.254.169.254 and 169.254.170.2` do not go through the HTTP proxy.
- ECS Exec and Amazon VPC
  - If you are using interface Amazon VPC endpoints with Amazon ECS, you must create the interface Amazon VPC endpoints for the Systems Manager Session Manager (`ssmmessages`). For more information about Systems Manager VPC endpoints, see Use AWS PrivateLink to set up a VPC endpoint for Session Manager in the *AWS Systems Manager User Guide*.
  - If you are using interface Amazon VPC endpoints with Amazon ECS, and you are using AWS KMS key for encryption, then you must create the interface Amazon VPC endpoint for AWS KMS key. For more information, see Connecting to AWS KMS key through a VPC endpoint in the *AWS Key Management Service Developer Guide*.
  - When you have tasks that run on Amazon EC2 instances, use `awsvpc` networking mode. If you don't have internet access (such as not configured to use a NAT gateway), you must create the interface Amazon VPC endpoints for the Systems Manager Session Manager (`ssmmessages`). For more information about `awsvpc` network mode considerations, see Considerations. For more information about Systems Manager VPC endpoints, see Use AWS PrivateLink to set up a VPC endpoint for Session Manager in the *AWS Systems Manager User Guide*.
- ECS Exec and SSM
  - When a user runs commands on a container using ECS Exec, these commands are run as the `root` user. The SSM agent and its child processes run as root even when you specify a user ID for the container.
  - The SSM agent requires that the container file system can be written to in order to create the required directories and files. Therefore, making the root file system read-only using the `readonlyRootFilesystem` task definition parameter, or any other method, isn't supported.
  - While starting SSM sessions outside of the `execute-command` action is possible, this results in the sessions not being logged and being counted against the session limit. We recommend

limiting this access by denying the `ssm:start-session` action using an IAM policy. For more information, see Limiting access to the Start Session action.

- The following features run as a sidecar container. Therefore, you must specify the container name to run the command on.
  - Runtime Monitoring
  - Service Connect
- Users can run all of the commands that are available within the container context. The following actions might result in orphaned and zombie processes: terminating the main process of the container, terminating the command agent, and deleting dependencies. To cleanup zombie processes, we recommend adding the `initProcessEnabled` flag to your task definition.
- ECS Exec uses some CPU and memory. You'll want to accommodate for that when specifying the CPU and memory resource allocations in your task definition.
- You must be using AWS CLI version `1.22.3` or later or AWS CLI version `2.3.6` or later. For information about how to update the AWS CLI, see Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide Version 2*.
- You can have only one ECS Exec session per process ID (PID) namespace. If you are sharing a PID namespace in a task, you can only start ECS Exec sessions into one container.
- The ECS Exec session has an idle timeout time of 20 minutes. This value can't be changed.
- You can't turn on ECS Exec for existing tasks. It can only be turned on for new tasks.
- You can't use ECS Exec when you use `run-task` to launch a task on a cluster that uses managed scaling with asynchronous placement (launch a task with no instance).
- You can't run ECS Exec against Microsoft Nano Server containers.

Architecture

ECS Exec makes use of AWS Systems Manager (SSM) Session Manager to establish a connection with the running container and uses AWS Identity and Access Management (IAM) policies to control access to running commands in a running container. This is made possible by bind-mounting the necessary SSM agent binaries into the container. The Amazon ECS or AWS Fargate agent is responsible for starting the SSM core agent inside the container alongside your application code. For more information, see Systems Manager Session Manager.

You can audit which user accessed the container using the `ExecuteCommand` event in AWS CloudTrail and log each command (and their output) to Amazon S3 or Amazon CloudWatch Logs.

To encrypt data between the local client and container with your own encryption key, you must provide the AWS Key Management Service (AWS KMS) key.

Configuring ECS Exec

To use ECS Exec, you must first turn on the feature for your tasks and services, and then you can run commands in your containers.

Optional task definition changes

If you set the task definition parameter `initProcessEnabled` to `true`, this starts the init process inside the container. This removes any zombie SSM agent child processes found. The following provides an example.

```
{
    "taskRoleArn": "ecsTaskRole",
    "networkMode": "awsvpc",
    "requiresCompatibilities": [
        "EC2",
        "FARGATE"
    ],
    "executionRoleArn": "ecsTaskExecutionRole",
    "memory": ".5 gb",
    "cpu": ".25 vcpu",
    "containerDefinitions": [
        {
            "name": "amazon-linux",
            "image": "amazonlinux:latest",
            "essential": true,
            "command": ["sleep","3600"],
            "linuxParameters": {
                "initProcessEnabled": true
            }
        }
    ],
    "family": "ecs-exec-task"
}
```

Turning on ECS Exec for your tasks and services

You can turn on the ECS Exec feature for your services and standalone tasks by specifying the `--enable-execute-command` flag when using one of the following AWS CLI commands: create-service, update-service, start-task, or run-task.

For example, if you run the following command, the ECS Exec feature is turned on for a newly created service that runs on Fargate. For more information about creating services, see create-service.

```
aws ecs create-service \
    --cluster cluster-name \
    --task-definition task-definition-name \
    --enable-execute-command \
    --service-name service-name \
    --launch-type FARGATE \
     --network-configuration
 "awsvpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=EN
 \
    --desired-count 1
```

After you turn on ECS Exec for a task, you can run the following command to confirm the task is ready to be used. If the `lastStatus` property of the `ExecuteCommandAgent` is listed as `RUNNING` and the `enableExecuteCommand` property is set to `true`, then your task is ready.

```
aws ecs describe-tasks \
    --cluster cluster-name \
    --tasks task-id
```

The following output snippet is an example of what you might see.

```
{
    "tasks": [
        {
            ...
            "containers": [
                {
                    ...
                    "managedAgents": [
                        {
```

```
                              "lastStartedAt": "2021-03-01T14:49:44.574000-06:00",
                              "name": "ExecuteCommandAgent",
                              "lastStatus": "RUNNING"
                          }
                      ]
                  }
              ],
              ...
              "enableExecuteCommand": true,
              ...
          }
      ]
}
```

Running commands using ECS Exec

Logging using ECS Exec

You can configure logging for ECS Exec sessions to capture commands and their output for auditing and troubleshooting purposes.

Turning on logging in your tasks and services

> **Important**
>
> For more information about CloudWatch pricing, see CloudWatch Pricing. Amazon ECS also provides monitoring metrics that are provided at no additional cost. For more information, see Monitor Amazon ECS using CloudWatch.

Amazon ECS provides a default configuration for logging commands run using ECS Exec. The default is to send logs to CloudWatch Logs using the `awslogs` log driver that's configured in your task definition. If you want to provide a custom configuration, the AWS CLI supports a `--configuration` flag for both the `create-cluster` and `update-cluster` commands. The container image requires `script` and `cat` to be installed in order to have command logs uploaded correctly to Amazon S3 or CloudWatch Logs. For more information about creating clusters, see create-cluster.

Note

This configuration only handles the logging of the `execute-command` session. It doesn't affect logging of your application.

The following example creates a cluster and then logs the output to your CloudWatch Logs LogGroup named `cloudwatch-log-group-name` and your Amazon S3 bucket named `s3-bucket-name`.

You must use an AWS KMS customer managed key to encrypt the log group when you set the `CloudWatchEncryptionEnabled` option to `true`. For information about how to encrypt the log group, see [Encrypt log data in CloudWatch Logs using AWS Key Management Service](), in the *Amazon CloudWatch Logs User Guide*.

```
aws ecs create-cluster \
    --cluster-name cluster-name \
    --configuration executeCommandConfiguration="{ \
        kmsKeyId=string, \
        logging=OVERRIDE, \
        logConfiguration={ \
            cloudWatchLogGroupName=cloudwatch-log-group-name, \
            cloudWatchEncryptionEnabled=true, \
            s3BucketName=s3-bucket-name, \
            s3EncryptionEnabled=true, \
            s3KeyPrefix=demo \
        } \
    }"
```

The `logging` property determines the behavior of the logging capability of ECS Exec:

- `NONE`: logging is turned off.
- `DEFAULT`: logs are sent to the configured `awslogs` driver. If the driver isn't configured, then no log is saved.
- `OVERRIDE`: logs are sent to the provided Amazon CloudWatch Logs LogGroup, Amazon S3 bucket, or both.

IAM permissions required for Amazon CloudWatch Logs or Amazon S3 Logging

To enable logging, the Amazon ECS task role that's referenced in your task definition needs to have additional permissions. These additional permissions can be added as a policy to the task role. They're different depending on if you direct your logs to Amazon CloudWatch Logs or Amazon S3.

Amazon CloudWatch Logs

The following example policy adds the required Amazon CloudWatch Logs permissions.

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
               "logs:CreateLogStream",
               "logs:DescribeLogStreams",
               "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:/aws/
ecs/cloudwatch-log-group-name:*"
        }
   ]
}
```

Amazon S3

The following example policy adds the required Amazon S3 permissions.

```
{
   "Version": "2012-10-17",
   "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
               "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
           "Effect": "Allow",
           "Action": [
               "s3:GetEncryptionConfiguration"
           ],
           "Resource": "arn:aws:s3:::s3-bucket-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::s3-bucket-name/*"
        }
    ]
  }
```

IAM permissions required for encryption using your own AWS KMS key (KMS key)

By default, the data transferred between your local client and the container uses TLS 1.2 encryption that AWS provides. To further encrypt data using your own KMS key, you must create a KMS key and add the `kms:Decrypt` permission to your task IAM role. This permission is used by your container to decrypt the data. For more information about creating a KMS key, see [Creating keys](#).

You add the following inline policy to your task IAM role which requires the AWS KMS permissions. For more information, see [ECS Exec permissions](#).

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-
east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

For the data to be encrypted using your own KMS key, the user or group using the `execute-command` action must be granted the `kms:GenerateDataKey` permission.

The following example policy for your user or group contains the required permission to use your own KMS key. You must specify the Amazon Resource Name (ARN) of your KMS key.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-
east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

Using IAM policies to limit access to ECS Exec

You limit user access to the execute-command API action by using one or more of the following IAM policy condition keys:

- `aws:ResourceTag/`*`clusterTagKey`*
- `ecs:ResourceTag/`*`clusterTagKey`*

- aws:ResourceTag/*taskTagKey*
- ecs:ResourceTag/*taskTagKey*
- ecs:container-name
- ecs:cluster
- ecs:task
- ecs:enable-execute-command

With the following example IAM policy, users can run commands in containers that are running within tasks with a tag that has an `environment` key and `development` value and in a cluster that's named `cluster-name`.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ExecuteCommand",
                "ecs:DescribeTasks"
            ],
            "Resource": [
                   "arn:aws:ecs:us-east-1:111122223333:task/cluster-name/*",
                   "arn:aws:ecs:us-east-1:111122223333:cluster/cluster-name"
            ],
            "Condition": {
                "StringEquals": {
                    "ecs:ResourceTag/environment": "development"
                }
            }
        }
    ]
}
```

With the following IAM policy example, users can't use the `execute-command` API when the container name is `production-app`.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "ecs:ExecuteCommand"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:container-name": "production-app"
                }
            }
        }
    ]
}
```

With the following IAM policy, users can only launch tasks when ECS Exec is turned off.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask",
                "ecs:StartTask",
                "ecs:CreateService",
                "ecs:UpdateService"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:enable-execute-command": "false"
                }
            }
```

```
        }
    ]
}
```

> **Note**
>
> Because the `execute-command` API action contains only task and cluster resources in a request, only cluster and task tags are evaluated.

For more information about IAM policy condition keys, see Actions, resources, and condition keys for Amazon Elastic Container Service in the *Service Authorization Reference*.

Limiting access to the Start Session action

While starting SSM sessions on your container outside of ECS Exec is possible, this could potentially result in the sessions not being logged. Sessions started outside of ECS Exec also count against the session quota. We recommend limiting this access by denying the `ssm:start-session` action directly for your Amazon ECS tasks using an IAM policy. You can deny access to all Amazon ECS tasks or to specific tasks based on the tags used.

The following is an example IAM policy that denies access to the `ssm:start-session` action for tasks in all Regions with a specified cluster name. You can optionally include a wildcard with the *`cluster-name`*.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "ssm:StartSession",
            "Resource": [
                  "arn:aws:ecs:us-east-1:111122223333:task/cluster-name/*"
            ]
        }
    ]
}
```

The following is an example IAM policy that denies access to the `ssm:start-session` action on resources in all Regions tagged with tag key `Task-Tag-Key` and tag value `Exec-Task`.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "ssm:StartSession",
            "Resource": "arn:aws:ecs:*:*:task/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Task-Tag-Key": "Exec-Task"
                }
            }
        }
    ]
}
```

Troubleshooting ECS Exec

For additional troubleshooting help, see Troubleshooting issues with Exec.

Running commands using ECS Exec

You can use Amazon ECS Exec to collect diagnostic information related to your containers and troubleshoot errors that are encountered throughout the lifecycle of your containers.

Prerequisites

Before you start using ECS Exec, make sure that you have completed these actions:

- Review the considerations. For more information, see Considerations
- Configure ECS Exec for your tasks and services. For more information, see Configuring ECS Exec
- **Install and configure the AWS CLI**. For more information, see Get started with the AWS CLI.
- **Install Session Manager plugin for the AWS CLI**. For more information, see Install the Session Manager plugin for the AWS CLI.

- **Configure a task role with appropriate permissions**. You must use a task role with the appropriate permissions for ECS Exec. For more information, see Task IAM role.
- **Verify version requirements**. ECS Exec has version requirements depending on whether your tasks are hosted on Amazon EC2 or AWS Fargate:
  - If you're using Amazon EC2, you must use an Amazon ECS optimized AMI that was released after January 20th, 2021, with an agent version of 1.50.2 or greater. For more information, see Amazon ECS optimized AMIs.
  - If you're using AWS Fargate, you must use platform version `1.4.0` or higher (Linux) or `1.0.0` (Windows). For more information, see AWS Fargate platform versions.

Using the console for service tasks

You can use the console to run commands using ECS Exec.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Services** section, choose the service.

   The service details page displays.
4. On the service details page, choose **Tasks**. Then, choose the task.
5. Under **Containers**, choose the container where you want to use ECS Exec.
6. To run commands:, do one of the following:
   - Choose **Connect**.

     A CloudShell session displays where you can run your commands.
   - Choose the arrow, and then choose **Copy AWS CLI command**.

     You can then run the commands locally.

Expected results

If the connection is successful, you should see an interactive shell prompt from your container. You can now run commands directly in the container environment. To exit the session, choose **End Session**.

Using the console for standalone tasks

You can use the console to run commands using ECS Exec.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster.
3. On the cluster details page, in the **Tasks** section, choose the task.

   The task details page displays.
4. Under **Containers**, choose the container where you want to use ECS Exec.
5. To run commands:, do one of the following:

   - Choose **Connect**.

     A CloudShell session displays where you can run your commands.
   - Choose the arrow, and then choose **Copy AWS CLI command**.

     You can then run the commands locally.

Expected results

If the connection is successful, you should see an interactive shell prompt from your container. You can now run commands directly in the container environment. To exit the session, choose **End Session**.

Using the command shell

You can use the command shell to run commands using ECS Exec.

After you have confirmed the `ExecuteCommandAgent` is running, you can open an interactive shell on your container using the following command. If your task contains multiple containers, you must specify the container name using the `--container` flag. Amazon ECS only supports initiating interactive sessions, so you must use the `--interactive` flag.

The following command will run an interactive `/bin/sh` command against a container named *`container-name`* for a task with an ID of *`task-id`*.

The *`task-id`* is the Amazon Resource Name (ARN) of the task.

```
aws ecs execute-command --cluster cluster-name \
```

```
    --task task-id \
    --container container-name \
    --interactive \
    --command "/bin/sh"
```

Expected results

If the command is successful, you should see an interactive shell prompt from your container. You can now run commands directly in the container environment. To exit the session, type `exit` or press `Ctrl+D`.

AWS Compute Optimizer recommendations for Amazon ECS

AWS Compute Optimizer generates recommendations for Amazon ECS task and container sizes. For more information, see [What is AWS Compute Optimizer?](#) in the *AWS Compute Optimizer User Guide*.

Task and container size recommendations for Amazon ECS services on AWS Fargate

AWS Compute Optimizer generates recommendations for Amazon ECS services on AWS Fargate. AWS Compute Optimizer recommends task CPU and task memory size and container CPU, container memory and container memory reservation sizes. These recommendations are displayed on the following pages of the Compute Optimizer console.

- **Recommendations for Amazon ECS services on Fargate** page
- **Amazon ECS services on Fargate details** page

For more information, see [Viewing recommendations for Amazon ECS services on Fargate](#) in the *AWS Compute Optimizer User Guide*.

Amazon ECS troubleshooting

You might need to troubleshoot issues with your load balancers, tasks, services, or container instances. This chapter helps you find diagnostic information from the Amazon ECS container agent, the Docker daemon on the container instance, and the service event log in the Amazon ECS console.

For information about stopped tasks, see one of the following.

Action	Learn more
Resolve stopped task errors.	Viewing Amazon ECS stopped task errors
View stopped task errors.	Resolve Amazon ECS stopped task errors
Review stopped task error codes.	Amazon ECS stopped tasks error messages
Review CannotPullContainer task errors.	CannotPullContainer task errors in Amazon ECS
View task IAM role requests.	Viewing IAM role requests for Amazon ECS tasks

For information about service errors, see one of the following.

Action	Learn more
View service event messages.	Viewing Amazon ECS service event messages
Review service event messages.	Amazon ECS service event messages

Action	Learn more
Review load balancer issues.	Troubleshooting service load balancers in Amazon ECS
Review service auto scaling issues.	Troubleshooting service auto scaling in Amazon ECS

For information about task definition errors, see one of the following.

Action	Learn more
Resolve task definition memory error.	Troubleshoot Amazon ECS task definition invalid CPU or memory errors

For information about Amazon ECS agent errors, see one of the following.

Action	Learn more
View Amazon ECS container agent logs.	Viewing Amazon ECS container agent logs
Learn how to collect Amazon ECS logs.	Collecting container logs with Amazon ECS logs collector
Retrieve diagnostic details with the Amazon ECS agent.	Retrieve Amazon ECS diagnostic details with agent introspection

For information about Docker errors, see one of the following.

Action	Learn more
Use Docker diagnostics.	Docker diagnostics in Amazon ECS
Turn on Docker debug mode.	Configuring verbose output from the Docker daemon in Amazon ECS
Troubleshoot Docker API error 500.	Troubleshoot the Docker `API error (500): devmapper` in Amazon ECS

For information about ECS Exec and Amazon ECS Anywhere errors, see one of the following.

Action	Learn more
Troubleshoot ECS Exec.	Troubleshoot Amazon ECS Exec issues
Troubleshoot Amazon ECS Anywhere.	Troubleshoot Amazon ECS Anywhere issues

For information about throttling issues, see one of the following.

Action	Learn more
Learn about Fargate throttlin g quotas.	AWS Fargate throttling quotas
Learn the best practices for Amazon ECS throttling.	Handle Amazon ECS throttlin g issues

For information about API errors, see one of the following.

Action	Learn more	
Resolve API errors.	Amazon ECS API failure reasons	

Resolve Amazon ECS stopped task errors

When your task fails to start, you see an error message in the console and in the `describe-tasks` output parameters (`stoppedReason` and `stopCode`).

You can view stopped tasks in the console for one hour. In order to see stopped tasks, you must change the filter option. For more information, see Viewing Amazon ECS stopped task errors.

The following pages provide information about stopped tasks.

- Learn about changes to stopped task error messages.

  Amazon ECS stopped task error messages updates
- View your stopped tasks so you can get information about the cause.

  Viewing Amazon ECS stopped task errors
- Learn about the stopped tasks error messages and possible reasons for the errors.

  Amazon ECS stopped tasks error messages
- Learn how to verify stopped task connectivity and fix the errors.

  Verifying Amazon ECS stopped task connectivity

Amazon ECS stopped task error messages updates

Beginning June 14, 2024 the Amazon ECS team is changing the stopped task error messages as described in the following tables. The `stopCode` will not change. If your applications depend on exact error message strings, you must update your applications with the new strings. For help with questions or problems, contact AWS Support.

> **Note**
>
> We recommend that you do not rely on the error messages for your automation, because the error messages are subject to change.

CannotPullContainerError

Old error message	New error message	
CannotPullContainerError: Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login': denied: User: *roleARN*	• CannotPullContainerError: The task can't pull the image. Check that the role has the permissions to pull images from the registry. Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login': denied: User: *roleARN* is not authorized to perform: ecr:BatchGetImage on resource: *image* because no identity-based policy allows the ecr:BatchGetImage action. • CannotPullContainerError: The task can't pull the image. Check whether the image exists. Error response from daemon: pull access denied for *repository* , repository does not exist or may require 'docker login':	

Old error message	New error message
	denied: requested access to the resource is denied.
CannotPullContainerError: Error response from daemon: Get *imageURI*: net/http: request canceled while waiting for connection	CannotPullContainerError: The task can’t pull the image. Check your network configura tion. Error response from daemon: Get *image*: net/ http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
CannotPullContainerError: ref pull has been retried 5 time(s): failed to copy: httpReadSeeker: failed open: failed to do request: Get *registry-uri* : dial tcp <ip>:<port> i/o timeout	CannotPullContainerError: The task cannot pull *image-uri* from the registry *registry-uri* . There is a connection issue between the task and the registry. Check your task network configuration. : failed to copy: httpReadSeeker: failed open: failed to do request: Get *registry-uri* : dial tcp <ip>:<port> i/o timeout

ResourceNotFoundException

Old error message	New error message
Fetching secret data from AWS Secrets Manager in *region* region: secret *sercretARN* : ResourceN otFoundException: Secrets	ResourceNotFoundException: The task can't retrieve the secret with ARN '*sercretAR N* ' from AWS Secrets Manager. Check whether the

Old error message	New error message
Manager can't find the specified secret.	secret exists in the specified Region. ResourceNotFoundEx ception: Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN* : ResourceN otFoundException: Secrets Manager can't find the specified secret.

ResourceInitializationError

Old error message	New error message
ResourceInitializationError : unable to pull secrets or registry auth: execution resource retrieval failed: unable to retrieve ecr registry auth: service call has been retried 3 time(s): RequestEr ror: send request failed caused by: Post "https:// api.ecr.us-east-1.amazonaws .com/": dial tcp <ip>:<port>: i/o timeout. Please check your task network configuration.	ResourceInitializationError : unable to pull secrets or registry auth: The task cannot pull registry auth from Amazon ECR: There is a connection issue between the task and Amazon ECR. Check your task network configuration. RequestError: send request failed caused by: Post "https://api.ecr.us-east-1.amazonaws.com": dial tcp <ip>:<port>: i/o timeout
ResourceInitializationError : unable to pull secrets or registry auth: execution resource retrieval failed: unable to retrieve secrets from ssm: service call has	ResourceInitializationError : unable to pull secrets or registry auth: unable to retrieve secrets from ssm: The task cannot pull secrets from AWS Systems Manager. There

Old error message	New error message
been retried 5 time(s): RequestCanceled: request context canceled caused by: context deadline exceeded	is a connection issue between the task and AWS Systems Manager Parameter Store. Check your task network configuration. RequestCa nceled: request context canceled caused by: context deadline exceeded
ResourceInitializationError: failed to download env files: file download command: non empty error stream: RequestCanceled: request context canceled caused by: context deadline exceeded	ResourceInitializationError: failed to download env files: The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration. service call has been retried 5 time(s): RequestCanceled: request context canceled caused by: context deadline exceeded
ResourceInitializationError : failed to validate logger args::signal:killed	ResourceInitializationError : failed to validate logger args: The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connection issue between the task and Amazon CloudWatc h. Check your network configuration. : signal: killed

Old error message	New error message	
ResourceInitializationError : unable to pull secrets or registry auth: pull command failed: : signal: killed	ResourceInitializationError : unable to pull secrets or registry auth: Check your task network configuration. : signal: killed	

Viewing Amazon ECS stopped task errors

If you have trouble starting a task, your task might be stopping because of application or configuration errors. For example, you run the task and the task displays a `PENDING` status and then disappears.

If your task was created by an Amazon ECS service, the actions that Amazon ECS takes to maintain the service are published in the service events. You can view the events in the AWS Management Console, AWS CLI, AWS SDKs, the Amazon ECS API, or tools that use the SDKs and API. These events include Amazon ECS stopping and replaces a task because the containers in the task have stopped running, or have failed too many health checks from Elastic Load Balancing.

If your task ran on a container instance on Amazon EC2 or external computers, you can also look at the logs of the container runtime and the Amazon ECS Agent. These logs are on the host Amazon EC2 instance or external computer. For more information, see Viewing Amazon ECS container agent logs.

Procedure

Console

AWS Management Console

The following steps can be used to check stopped tasks for errors using the console. In order to see stopped tasks, you must change the filter option.

Stopped tasks only appear in the console for 1 hour.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.

3. On the **Clusters** page, choose the cluster.
4. On the **Cluster :** ***name*** page, choose the **Tasks** tab.
5. Configure the filter to display stopped tasks. For **Filter desired status**, choose **Stopped**.

   The **Stopped** option displays your stopped tasks and **Any desired status** displays all of your tasks.
6. Choose the stopped task to inspect.
7. In the row for your stopped task, in the **Last Status** column, choose **Stopped**.

   A pop-up window displays the stopped reason.

AWS CLI

1. List the stopped tasks in a cluster. The output contains the Amazon Resource Name (ARN) of the task, which you need to describe the task.

```
aws ecs list-tasks \
     --cluster cluster_name \
     --desired-status STOPPED \
     --region region
```

2. Describe the stopped task to retrieve the information. For more information, see describe-tasks in the *AWS Command Line Interface Reference*.

```
aws ecs describe-tasks \
     --cluster cluster_name \
     --tasks arn:aws:ecs:region:account_id:task/cluster_name/task_ID \
     --region region
```

Use the following output parameters.

- `stopCode` - The stop code indicates why a task was stopped, for example ResourceInitializationError
- `StoppedReason` - The reason the task stopped.
- `reason` (in the `containers` structure) - The reason provide additional details about the stopped container.

Next steps

View your stopped tasks so you can get information about the cause. For more information, see [Amazon ECS stopped tasks error messages].

Amazon ECS stopped tasks error messages

The following are the possible error messages you may receive when your task stops unexpectedly.

To check your stopped tasks for an error message using the AWS Management Console, see [Viewing Amazon ECS stopped task errors].

Stopped task error codes have a category associated with them, for example "ResourceInitializationError". To get more information about each category, see the following:

Category	Learn more	
TaskFailedToStart	Troubleshooting Amazon ECS TaskFailedToStart errors	
ResourceInitializationError	Troubleshooting Amazon ECS ResourceInitializationError errors	
ResourceNotFoundException	Troubleshooting Amazon ECS ResourceNotFoundException errors	
SpotInterruptionError	Troubleshooting Amazon ECS SpotInterruption errors	
InternalError	Troubleshooting Amazon ECS InternalError errors	
OutOfMemoryError	Troubleshooting Amazon ECS OutOfMemoryError errors	
ContainerRuntimeError	Troubleshooting Amazon ECS ContainerRuntimeError errors	

Category	Learn more
ContainerRuntimeTi meoutError	Troubleshooting Amazon ECS ContainerRuntimeTi meoutError errors
CannotStartContainerError	Troubleshooting Amazon ECS CannotStartContainerError errors
CannotStopContainerError	Troubleshooting Amazon ECS CannotStopContainerError errors
CannotInspectContainerError	Troubleshooting Amazon ECS CannotInspectContainerError errors
CannotCreateVolumeError	Troubleshooting Amazon ECS CannotCreateVolumeError errors
CannotPullContainer	CannotPullContainer task errors in Amazon ECS

Troubleshooting Amazon ECS TaskFailedToStart errors

The following are some `TaskFailedToStart` error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

Unexpected EC2 error while attempting to Create Network Interface with public IP assignment enabled in subnet '*`subnet-id`*

This happens when a Fargate task that uses the `awsvpc` network mode and runs in a subnet with a public IP address, and the subnet does not have enough IP addresses.

The number of available IP addresses is available on the subnet details page in the Amazon EC2 console, or by using `describe-subnets`. For more information, see View your subnet in the *Amazon VPC User Guide*.

To fix this issue, you can create a new subnet to run your task in.

InternalError: *<reason>*

This error occurs when an ENI attachment is requested. Amazon EC2 asynchronously handles the provisioning of the ENI. The provisioning process takes time. Amazon ECS has a timeout in case there are long wait times or unreported failures. There are times when the ENI is provisioned, but the report comes to Amazon ECS after the failure timeout. In this case, Amazon ECS sees the reported task failure with an in-use ENI.

The selected task definition is not compatible with the selected compute strategy

This error occurs when you chose a task definition with a launch type that does not match the cluster capacity type. For more information, see Amazon ECS launch types. You need to select a task definition that matches the capacity provider assigned to your cluster.

Unable to attach network interface to unused device index

This error occurs when When using `awsvpc` networking type and there is not enough CPU/memory for the task. First, check the CPU for the instances. For more information, see Amazon EC2 instance type specifications in *Amazon EC2 instance types*. Take the CPU value for the instance and multiply it by the number of ENIs for the instance. Use that value e in the task definition.

AGENT

The container instance that you attempted to launch a task onto has an agent that's currently disconnected. To prevent extended wait times for task placement, the request was rejected.

For information about how to troubleshoot an agent that's disconnected, see How do I troubleshoot a disconnected Amazon ECS agent.

Troubleshooting Amazon ECS ResourceInitializationError errors

The following are some `ResourceInitialization` error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

Errors

- The task cannot pull registry authentication from Amazon ECR. There is a connection issue between the task and Amazon ECR. Check your task network configuration.
- The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration.
- The task cannot pull secrets from AWS Systems Manager Parameter Store. Check your network connection between the task and AWS Systems Manager.
- The task can’t pull secrets from AWS Secrets Manager. There is a connection issue between the task and Secrets Manager. Check your task network configuration.
- The task can’t pull the secret from Secrets Manager. The task can't retrieve the secret with ARN ‘secretARN' from Secrets Manager. Check whether the secret exists in the specified Region.
- pull command failed: unable to pull secrets or registry auth Check your task network configuration.
- The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connection issue between the task and Amazon CloudWatch. Check your network configuration.
- failed to initialize logging driver
- failed to invoke EFS utils commands to set up EFS volumes

The task cannot pull registry authentication from Amazon ECR. There is a connection issue between the task and Amazon ECR. Check your task network configuration.

This error indicates that the task can't connect to Amazon ECR.

Check the connection between the task and Amazon ECR. For information, see Verifying Amazon ECS stopped task connectivity.

The task can't download the environment variable files from Amazon S3. There is a connection issue between the task and Amazon S3. Check your task network configuration.

This error occurs when your task can't download your environment file from Amazon S3.

Check the connection between the task and the Amazon S3 VPC endpoint. For information, see Verifying Amazon ECS stopped task connectivity.

The task cannot pull secrets from AWS Systems Manager Parameter Store. Check your network connection between the task and AWS Systems Manager.

This error occurs when your task can't pull the image defined in the task definition using the credentials in Systems Manager.

Check the connection between the task and the Systems Manager VPC endpoint. For information, see Verifying Amazon ECS stopped task connectivity.

The task can’t pull secrets from AWS Secrets Manager. There is a connection issue between the task and Secrets Manager. Check your task network configuration.

This error occurs when your task can't pull the image defined in the task definition using the credentials in Secrets Manager.

The error indicates that there is a network connectivity issue between the Systems Manager VPC endpoint and the task.

For information about how to verify the connectivity between the task and the endpoint, see Verifying Amazon ECS stopped task connectivity.

The task can’t pull the secret from Secrets Manager. The task can't retrieve the secret with ARN ‘*secretARN*' from Secrets Manager. Check whether the secret exists in the specified Region.

This error occurs when your task can't pull the image defined in the task definition using the credentials in Secrets Manager.

This issue is caused by one of the following reasons:

Error cause..	Do this...	
Network connectivity issue between the Secrets Manager VPC endpoint and the task. The problem is a network issue when you see any of the following strings in the error message:	Verify the connectivity between the task and the Secrets Manager endpoint. For more information, see Verifying Amazon ECS stopped task connectivity.	

Error cause..	Do this...	
• dial tcp • dial udp • <ip>:<port>: i/o timeout • net/http: TLS handshake timeout • read: connection timed out • Client.Timeout exceeded while awaiting headers • net/http: request canceled while waiting for connectio n • signal: killed • context deadline exceeded		
The task execution role defined in the task definition doesn't have the permissions for Secrets Manager.	Add the required permissio ns for Secrets Manager to the task execution role. For more information, see [Secrets Manager or Systems Manager permissions]().	
The secret ARN doesn't exist	Check that the ARN exists in Secrets Manager. For information about viewing your images, see [Find secrets in Secrets Manager]() in the *Secrets Manager Developer Guide*.	

pull command failed: unable to pull secrets or registry auth Check your task network configuration.

This error occurs when your task can't connect to Amazon ECR, Systems Manager, or Secrets Manager. This is due to a misconfiguration in your network.

To fix this issue, verify the connectivty between the task and Amazon ECR. You also need to check connectivity between your task and the service which stores your secret (Systems Manager, or Secrets Manager). For more information, see Verifying Amazon ECS stopped task connectivity.

The task cannot find the Amazon CloudWatch log group defined in the task definition. There is a connection issue between the task and Amazon CloudWatch. Check your network configuration.

This error occurs when your task fails to find the CloudWatch log group you defined in the task definition.

The error indicates that there is a network connectivity issue between the CloudWatch VPC endpoint and the task.

For information about how to verify the connectivity between the task and the endpoint, see Verifying Amazon ECS stopped task connectivity.

failed to initialize logging driver

This error occurs when your task fails to find the CloudWatch log group you defined in the task definition.

The error indicates that the CloudWatch group in the task definition does not exist.

Use the following steps to find the missing CloudWatch.

1. Run the following command to get the task definition information.

   ```
   aws ecs describe-task-definition \
       --task-definition task-def-name
   ```

   Look at the output for each container and note the `awslogs-group` value.

   ```
   "logConfiguration": {
                   "logDriver": "awslogs",
                   "options": {
                       "awslogs-group": "/ecs/example-group",
                       "awslogs-create-group": "true",
                       "awslogs-region": "us-east-1",
                       "awslogs-stream-prefix": "ecs"
                   },
   ```

2. Verify that the group existis in CloudWatch for more information, see Working with log groups and log streams in the *Amazon CloudWatch Logs User Guide*.

   The issue is either that the group specified in the task definition is incorrect, or the log group does not exist.

3. Fix the issue.

The issue is...	Do this...	
The incorrect log group is specified in the task definition.	Update the task definitio n to include the log group configuration in the container definition.For information about updating the task definition, see Updating an Amazon ECS task definition using the console or RegisterT askDefinition in the *Amazon Elastic Container Service API Reference*.	
The log group does not exist in CloudWatch	Create the log group. For more information, see Create a log group in CloudWatch Logs in the *Amazon CloudWatch Logs User Guide*.	

failed to invoke EFS utils commands to set up EFS volumes

The following issues might prevent you from mounting your Amazon EFS volumes on your asks:

- The Amazon EFS file system isn't configured correctly.
- The task doesn't have the required permissions.
- There are issues related to network and VPC configurations.

For information about how to debug and fix this issue, see Why can't I mount my Amazon EFS volumes on my AWS Fargate tasks on AWS re:Post.

Troubleshooting Amazon ECS ResourceNotFoundException errors

The following are some `ResourceNotFoundException` error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

The task can't retrieve the secret with ARN '*sercretARN*' from AWS Secrets Manager. Check whether the secret exists in the specified Region.

This error occurs when the task can't retrieve the secret from Secrets Manager. This means that the secret specified in the task definition (and contained in the error message) does not exist in Secrets Manager.

The Region is in the error message.

Fetching secret data from AWS Secrets Manager in region *region*: secret *sercretARN*: ResourceNotFoundException: Secrets Manager can't find the specified secret.

For information about finding a secret, see Find secrets in AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

Use the following table to determine and address the error.

Issue	Actions	
The secret is in a different Region from the the task definition.	a. Create the secret in the same Region as the task. For more information, see Create an AWS Secrets Manager secret. b. Update the task definitio n with the new secret. For more information, see Updating an Amazon ECS task definition using	

Issue	Actions
	the console or RegisterTaskDefinition in the *Amazon Elastic Container Service API Reference*.
The task definition has the incorrect secret ARN. The correct secret exists in Secrets Manager.	Update the task definition with the correct secret. For more information, see Updating an Amazon ECS task definition using the console or RegisterTaskDefinition in the *Amazon Elastic Container Service API Reference*.
The secret no longer exists.	a. Create the secret in the same Region as the task. For more information, see Create an AWS Secrets Manager secret. b. Update the task definition with the new secret. For more information, see Updating an Amazon ECS task definition using the console or RegisterTaskDefinition in the *Amazon Elastic Container Service API Reference*.

Troubleshooting Amazon ECS SpotInterruption errors

The `SpotInterruption` error has different reasons for the Fargate and EC2 launch types.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

Fargate launch type

The `SpotInterruption` error occurs when there is no Fargate Spot capacity or when Fargate takes back Spot capacity.

You can have your tasks run in multiple Availability Zones to allow for more capacity.

EC2 launch type

This error occurs when there are no available Spot Instances or EC2 takes back Spot Instance capacity.

You can have your instances run in multiple Availability Zones to allow for more capacity.

Troubleshooting Amazon ECS InternalError errors

Applies to: Fargate launch type

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

The `InternalError` error when the agent encounters an unexpected, non-runtime related internal error.

This error only occurs if using platform version `1.4` or later.

For information about how to debug and fix this issue, see Amazon ECS stopped tasks error messages.

Troubleshooting Amazon ECS OutOfMemoryError errors

The following are some OutOfMemoryError error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

container killed due to memory usage

This error occurs when a container exits due to processes in the container consuming more memory than was allocated in the task definition, or due to host or operating system constraints.

Troubleshooting Amazon ECS ContainerRuntimeError errors

The following are some ContainerRuntimeError error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see [Viewing Amazon ECS stopped task errors](#).

ContainerRuntimeError

This error occurs when the agent receives an unexpected error from `containerd` for a runtime-specific operation. This error is usually caused by an internal failure in the agent or the `containerd` runtime.

This error only occurs if you use platform version `1.4.0` or later (Linux) or `1.0.0` or later (Windows).

For information about how to debug and fix this issue, see [Why is my Amazon ECS task Stopped](#) on AWS re:Post.

Troubleshooting Amazon ECS ContainerRuntimeTimeoutError errors

The following are some ContainerRuntimeTimeoutError error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see [Viewing Amazon ECS stopped task errors](#).

Could not transition to running; timed out after waiting 1m or Docker timeout error

This error occurs when a container can't transition to either a `RUNNING` or `STOPPED` state within the timeout period. The reason and timeout value is provided in the error message.

Troubleshooting Amazon ECS CannotStartContainerError errors

The following are some CannotStartContainerError error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see [Viewing Amazon ECS stopped task errors](#).

failed to get container status: *<reason>*

This error occurs when a container can't be started.

If your container attempts to exceed the memory specified here, the container is stopped. Increase the memory presented to the container. This is the `memory` parameter in the task definition. For more information, see the section called "Memory".

Troubleshooting Amazon ECS CannotStopContainerError errors

The following are some CannotStopContainerError error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

CannotStopContainerError

This error occurs when a container can't be stopped.

For information about how to debug and fix this issue, see Why is my Amazon ECS task Stopped on AWS re:Post.

Troubleshooting Amazon ECS CannotInspectContainerError errors

The following are some CannotInspectContainerError error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

CannotInspectContainerError

This error occurs when the container agent can't describe the container through the container runtime.

When using platform version `1.3` or earlier, the Amazon ECS agent returns the reason from Docker.

When using platform version `1.4.0` or later (Linux) or `1.0.0` or later (Windows), the Fargate agent returns the reason from `containerd`.

For information about how to debug and fix this issue, see Why is my Amazon ECS task Stopped on AWS re:Post.

Troubleshooting Amazon ECS CannotCreateVolumeError errors

The following are some CannotCreateVolumeError error messages and actions that you can take to fix the errors.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

CannotCreateVolumeError

This error occurs when the agent can't create the volume mount specified in the task definition.

This error only occurs if you use platform version `1.4.0` or later (Linux) or `1.0.0` or later (Windows).

For information about how to debug and fix this issue, see Why is my Amazon ECS task Stopped on AWS re:Post.

CannotPullContainer task errors in Amazon ECS

The following errors indicate that the task failed to start because Amazon ECS can't retrieve the specified container image.

> **Note**
>
> The 1.4 Fargate platform version truncates long error messages.

To check your stopped tasks for an error message using the AWS Management Console, see Viewing Amazon ECS stopped task errors.

Errors

- The task can’t pull the image. Check that the role has the permissions to pull images from the registry.
- The task cannot pull ‘image-name’ from the Amazon ECR repository ‘repository URI’. There is a connection issue between the task and Amazon ECR. Check your task network configuration.
- The task can’t pull the image. Check your network configuration

- CannotPullContainerError: pull image manifest has been retried 5 time(s): failed to resolve ref
- API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection
- API error
- write /var/lib/docker/tmp/GetImageBlob111111111: no space left on device
- ERROR: toomanyrequests: Too Many Requests or You have reached your pull rate limit.
- Error response from daemon: Get url: net/http: request canceled while waiting for connection
- ref pull has been retried 1 time(s): failed to copy: httpReaderSeeker: failed open: unexpected status code
- pull access denied
- pull command failed: panic: runtime error: invalid memory address or nil pointer dereference
- error pulling image conf/error pulling image configuration
- Context canceled

The task can't pull the image. Check that the role has the permissions to pull images from the registry.

This error indicates that the task can't pull the image specified in the task definition because of permission issues.

To resolve this issue:

1. Check that the image exists in the *repository*. For information about viewing your images, see Viewing image details in Amazon ECR in the *Amazon Elastic Container Registry User Guide*.
2. Verify that the *role-arn* has the correct permissions to pull the image.

   For information about how to update roles, see Update permissions for a role in the *AWS Identity and Access Management Use Guide*.

   The task uses one of the following roles:

   - For tasks with the Fargate launch type, this is the task execution role. For information about the additional permissions for Amazon ECR, Fargate tasks pulling Amazon ECR images over interface endpoints permissions.
   - For tasks with the EC2 launch type, this is the container instance role. For information about the additional permissions for Amazon ECR, Amazon ECR permissions.

The task cannot pull '*image-name*' from the Amazon ECR repository '*repository URI*'. There is a connection issue between the task and Amazon ECR. Check your task network configuration.

This error indicates that the task can't connect to Amazon ECR. Check the connection to the *repository URI* repository.

For information about how to verify and resolve the issue, see Verifying Amazon ECS stopped task connectivity.

The task can't pull the image. Check your network configuration

This error indicates that the task can't connect to Amazon ECR.

For information about how to verify and resolve the issue, see Verifying Amazon ECS stopped task connectivity.

CannotPullContainerError: pull image manifest has been retried 5 time(s): failed to resolve ref

This error indicates that the task can't pull the image.

To resolve this, you can:

- Verify that the image specified in the task definition matches the image in the repository.
- Amazon ECS forces image version stability. If the original image is no longer available you get this error. The image tag is part of enforcing this behavior. Change the image in the task definition from using :latest as the tag to a specifc version. For more information, see Container image resolution.

For information about how to verify and resolve the issue, see Verifying Amazon ECS stopped task connectivity.

API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection

This error indicates that a connection timed out, because a route to the internet doesn't exist.

To resolve this issue, you can:

- For tasks in public subnets, specify **ENABLED** for **Auto-assign public IP** when launching the task. For more information, see Running an application as an Amazon ECS task.

- For tasks in private subnets, specify **DISABLED** for **Auto-assign public IP** when launching the task, and configure a NAT gateway in your VPC to route requests to the internet. For more information, see NAT Gateways in the *Amazon VPC User Guide*.

API error

This error indicates that there is a connection issue with the Amazon ECR endpoint.

For information about how to resolve this issue, see How can I resolve the Amazon ECR error "CannotPullContainerError: API error" in Amazon ECS on the Support website.

write /var/lib/docker/tmp/*GetImageBlob111111111*: no space left on device

This error indicates that there is insufficient disk space.

To resolve this issue, free up disk space.

If you are using the Amazon ECS-optimized AMI, you can use the following command to retrieve the 20 largest files on your file system:

```
du -Sh / | sort -rh | head -20
```

Example output:

```
5.7G    /var/lib/docker/
containers/50501b5f4cbf90b406e0ca60bf4e6d4ec8f773a6c1d2b451ed8e0195418ad0d2
1.2G    /var/log/ecs
594M    /var/lib/docker/devicemapper/mnt/
c8e3010e36ce4c089bf286a623699f5233097ca126ebd5a700af023a5127633d/rootfs/data/logs
...
```

In some cases, the root volume might be filled out by a running container. If the container is using the default `json-file` log driver without a `max-size` limit, it may be that the log file is responsible for most of that space used. You can use the `docker ps` command to verify which container is using the space by mapping the directory name from the output above to the container ID. For example:

```
CONTAINER ID   IMAGE                                                       COMMAND                CREATED
 STATUS                PORTS                                                       NAMES
```

```
50501b5f4cbf    amazon/amazon-ecs-agent:latest    "/agent"            4 days ago
 Up 4 days                                              ecs-agent
```

By default, when using the `json-file` log driver, Docker captures the standard output (and standard error) of all of your containers and writes them in files using the JSON format. You can set the `max-size` as a log driver option, which prevents the log file from taking up too much space. For more information, see JSON File logging driver in the Docker documentation.

The following is a container definition snippet showing how to use this option:

```
{
    "log-driver": "json-file",
    "log-opts": {
        "max-size": "256m"
    }
}
```

An alternative, if your container logs are taking up too much disk space, is to use the `awslogs` log driver. The `awslogs` log driver sends the logs to CloudWatch, which frees up the disk space that would otherwise be used for your container logs on the container instance. For more information, see Send Amazon ECS logs to CloudWatch .

You might need to update the disk size that Docker can access.

For more information, see CannotPullContainerError: no space left on device.

ERROR: toomanyrequests: Too Many Requests or You have reached your pull rate limit.

This error indicates that there is a Docker Hub rate limiting.

If you receive one of the following errors, you're likely hitting the Docker Hub rate limits:

For more information about the Docker Hub rate limits, see Understanding Docker Hub rate limiting.

If you have increased the Docker Hub rate limit and you need to authenticate your Docker pulls for your container instances, see Private registry authentication for container instances.

Error response from daemon: Get *`url`*: net/http: request canceled while waiting for connection

This error indicates that a connection timed out, because a route to the internet doesn't exist.

To resolve this issue, you can:

- For tasks in public subnets, specify **ENABLED** for **Auto-assign public IP** when launching the task. For more information, see Running an application as an Amazon ECS task.
- For tasks in private subnets, specify **DISABLED** for **Auto-assign public IP** when launching the task, and configure a NAT gateway in your VPC to route requests to the internet. For more information, see NAT Gateways in the *Amazon VPC User Guide*.

ref pull has been retried 1 time(s): failed to copy: httpReaderSeeker: failed open: unexpected status code

This error indicates that there was a failure when copying an image.

To resolve this issue, review one of the following articles:

- For Fargate tasks, see How do I resolve the "cannotpullcontainererror" error for my Amazon ECS tasks on Fargate.
- For other tasks, see How do I resolve the "cannotpullcontainererror" error for my Amazon ECS tasks.

pull access denied

This error indicates that there is no access to the image.

To resolve this issue, you might need to authenticate your Docker client with Amazon ECR For more information, see Private registry authentication in the *Amazon ECR User Guide*.

pull command failed: panic: runtime error: invalid memory address or nil pointer dereference

This error indicates that there is no access to the image because of an invalid memory address or nil pointer dereference.

To resolve this issue:

- Check that you have the security group rules to reach Amazon S3.
- When you use gateway endpoints, you must add a route in the route table to access the endpoint.

error pulling image conf/error pulling image configuration

This error indicates a rate limit has been reached or there is a network error:

To resolve this issue, see [How can I resolve the "CannotPullContainerError" error in my Amazon ECS EC2 Launch Type Task].

Context canceled

This error indicates that the context was cancelled.

The common cause for this error is because the VPC your task is using doesn't have a route to pull the container image from Amazon ECR.

Verifying Amazon ECS stopped task connectivity

There are times when a task stops because of a network connectivity issue. It might be an intermittent issue, but it is most likely caused because the task cannot connect to an endpoint.

Testing the task connectivity

You can use `AWSSupport-TroubleshootECSTaskFailedToStart` runbook to test the task connectivity. When you use the runbook, you need the following resource information:

- The task ID

  Use the ID of the most recent failed task.
- The cluster that the task was in

For information about how to use the runbook, see `AWSSupport-TroubleshootECSTaskFailedToStart` in the *AWS Systems Manager Automation runbook reference*.

The runbook analyzes the task. You can view the results in the **Output** section for the following issues that can prevent a task from starting:

- Network connectivity to the configured container registry
- VPC endpoint connectivity
- Security group rule configuration

Fixing VPC endpoint issues

When the `AWSSupport-TroubleshootECSTaskFailedToStart` runbook result indicates the VPC endpoint issue, check the following configuration:

- The VPC where you create the endpoint needs to use Private DNS.
- Make sure that you have a AWS PrivateLink endpoint for the service that the task cannot connect to in the same VPC as the task. For more information see one of the following:

Service	VPC endpoint information for the service	
Amazon ECR	Amazon ECR interface VPC endpoints (AWS PrivateLink)	
Systems Manager	Improve the security of EC2 instances by using VPC endpoints for Systems Manager	
Secrets Manager	Using an AWS Secrets Manager VPC endpoint	
CloudWatch	CloudWatch VPC endpoint	
Amazon S3	AWS PrivateLink for Amazon S3	

- Configure an outbound rule for the task subnet which allows HTTPS on port 443 DNS (TCP) traffic. For more information, see Configure security group rules in the *Amazon Elastic Compute Cloud User Guide*.
- If you use a custom name domain server, then confirm the DNS query's settings. The query must have outbound access on port 53, and use UDP and TCP protocol. Also, it must have HTTPS access on port 443. For more information, see Coonfigure security group rules in the *Amazon Elastic Compute Cloud User Guide*.
- If the subnet has a network ACL, the following ACL rules are required:
  - An outbound rule that allows traffic that allows traffic on ports 1024-65535.
  - An inbound rule that allows TCP traffic on port 443.

  For information about how to configure rules, see Control traffic to subnets using network ACLs in the *Amazon Virtual Private Cloud User Guide*.

Fixing network issues

When the `AWSSupport-TroubleshootECSTaskFailedToStart` runbook result indicates a network issue, check the following configuration:

Tasks that use awsvpc network mode in a public subnet

Perform the following configuration based on the runbook:

- For tasks in public subnets, specify **ENABLED** for **Auto-assign public IP** when launching the task. For more information, see Running an application as an Amazon ECS task.
- You need a gateway to handle internet traffic. The route table for the task subnet needs to have a route for traffic to the gateway.

  For more information, see Add and remove routes from a route table in the *Amazon Virtual Private Cloud User Guide*.

Gateway type	Route table destination	Rout table target
NAT	0.0.0.0/0	NAT gateway ID
Internet gateway	0.0.0.0/0	Internet gateway ID

- If the task subnet has a network ACL, the following ACL rules are required:
  - An outbound rule that allows traffic on ports 1024-65535.
  - An inbound rule that allows TCP traffic on port 443.

  For information about how to configure rules, see Control traffic to subnets using network ACLs in the *Amazon Virtual Private Cloud User Guide*.

Tasks that use awsvpc network mode in a private subnet

Perform the following configuration based on the runbook:

- Choose **DISABLED** for **Auto-assign public IP** when launching the task.
- Configure a NAT gateway in your VPC to route requests to the internet. For more information, see NAT Gateways in the *Amazon Virtual Private Cloud User Guide*.
- The route table for the task subnet needs to have a route for traffic to the NAT gateway.

For more information, see Add and remove routes from a route table in the *Amazon Virtual Private Cloud User Guide*.

Gateway type	Route table destination	Rout table target
NAT	0.0.0.0/0	NAT gateway ID

- If the task subnet has a network ACL, the following ACL rules are required:
  - An outbound rule that allows traffic on ports 1024-65535.
  - An inbound rule that allows TCP traffic on port 443.

  For information about how to configure rules, see Control traffic to subnets using network ACLs in the *Amazon Virtual Private Cloud User Guide*.

Tasks that don't use awsvpc network mode in a public subnet

Perform the following configuration based on the runbook:

- Choose **Turn on** for **Auto assign IP** under **Networking for Amazon EC2 instances** when you create the cluster.

  This option assigns a public IP address to the instance primary network interface.
- You need a gateway to handle internet traffic. The route table for the instance subnet needs to have a route for traffic to the gateway.

  For more information, see Add and remove routes from a route table in the *Amazon Virtual Private Cloud User Guide*.

Gateway type	Route table destination	Rout table target
NAT	0.0.0.0/0	NAT gateway ID
Internet gateway	0.0.0.0/0	Internet gateway ID

- If the instance subnet has a network ACL, the following ACL rules are required:
  - An outbound rule that allows traffic on ports 1024-65535.
  - An inbound rule that allows TCP traffic on port 443.

For information about how to configure rules, see Control traffic to subnets using network ACLs in the *Amazon Virtual Private Cloud User Guide*.

Tasks that don't use awsvpc network mode in a private subnet

Perform the following configuration based on the runbook:

- Choose **Turn off** for **Auto assign IP** under **Networking for Amazon EC2 instances** when you create the cluster.
- Configure a NAT gateway in your VPC to route requests to the internet. For more information, see NAT Gateways in the *Amazon VPC User Guide*.
- The route table for the instance subnet needs to have a route for traffic to the NAT gateway.

  For more information, see Add and remove routes from a route table in the *Amazon Virtual Private Cloud User Guide*.

Gateway type	Route table destination	Rout table target
NAT	0.0.0.0/0	NAT gateway ID

- If the task subnet has a network ACL, the following ACL rules are required:
  - An outbound rule that allows traffic on ports 1024-65535.
  - An inbound rule that allows TCP traffic on port 443.

  For information about how to configure rules, see Control traffic to subnets using network ACLs in the *Amazon Virtual Private Cloud User Guide*.

Viewing IAM role requests for Amazon ECS tasks

When you use a provider for your task credentials in an IAM role, the provider requests saved in an audit log. The audit log inherits the same log rotation settings as the container agent log. The `ECS_LOG_ROLLOVER_TYPE`, `ECS_LOG_MAX_FILE_SIZE_MB`, and `ECS_LOG_MAX_ROLL_COUNT` container agent configuration variables can be set to affect the behavior of the audit log. For more information, see Amazon ECS container agent log configuration parameters.

For container agent version 1.36.0 and later, the audit log is located at `/var/log/ecs/audit.log`. When the log is rotated, a timestamp in *YYYY*-*MM*-*DD*-*HH* format is added to the end of the log file name.

For container agent version 1.35.0 and earlier, the audit log is located at `/var/log/ecs/audit.log.`*YYYY*-*MM*-*DD*-*HH*.

The log entry format is as follows:

- Timestamp
- HTTP response code
- IP address and port number of request origin
- Relative URI of the credential provider
- The user agent that made the request
- The ARN of the task to which the requesting container belongs
- The `GetCredentials` API name and version number
- The name of the Amazon ECS cluster to which the container instance is registered
- The container instance ARN

You can use the following command to view the log files.

```
cat /var/log/ecs/audit.log.2016-07-13-16
```

Output:

```
2016-07-13T16:11:53Z 200 172.17.0.5:52444 "/v1/credentials" "python-requests/2.7.0
 CPython/2.7.6 Linux/4.4.14-24.50.amzn1.x86_64" TASK_ARN GetCredentials
 1 CLUSTER_NAME CONTAINER_INSTANCE_ARN
```

Viewing Amazon ECS service event messages

When troubleshooting a problem with a service, the first place you should check for the diagnostic information is the service event log. You can view service events using the `DescribeServices` API, the AWS CLI, or by using the AWS Management Console.

When viewing service event messages using the Amazon ECS API, only the events from the service scheduler are returned. These include the most recent task placement and instance health events. However, the Amazon ECS console displays service events from the following sources.

- Task placement and instance health events from the Amazon ECS service scheduler. These events have a prefix of **service *(service-name)***. To ensure that this event view is helpful, we only show the `100` most recent events. Duplicate event messages are omitted until either the cause is resolved, or six hours passes. If the cause is not resolved within six hours, you receive another service event message for that cause.
- Service Auto Scaling events. These events have a prefix of **Message** and occur only when a service is configured with an Application Auto Scaling scaling policy.

Use the following steps to view your current service event messages.

Console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.
4. Choose the service to inspect.
5. Choose **Deployments and events**, under **Events**, view the messages.

AWS CLI

Use the describe-services command to view the service event messages for a specified service.

The following AWS CLI example describes the *service-name* service in the *default* cluster, which will provide the latest service event messages.

```
aws ecs describe-services \
    --cluster default \
    --services service-name \
    --region us-west-2
```

Amazon ECS service event messages

The following are examples of service event messages you may see in the Amazon ECS console.

service (*service-name*) has reached a steady state.

The service scheduler sends a `service (service-name) has reached a steady state.` service event when the service is healthy and at the desired number of tasks, thus reaching a steady state.

The service scheduler reports the status periodically, so you might receive this message multiple times.

service (*service-name*) was unable to place a task because no container instance met all of its requirements.

The service scheduler sends this event message when it couldn't find the available resources to add another task. The possible causes for this are:

Use capacity providers to automatically scale your EC2 instances. For more information, see [Amazon ECS capacity providers for the EC2 launch type](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/asg-capacity-providers.html).

> If you intended to use a capacity provider, make sure that you're either passing a capacity provider strategy or have a default capacity provider strategy associated with your cluster and are not passing launch type and capacity provider strategy as input

No container instances were found in your cluster

> If no container instances are registered in the cluster you attempt to run a task in, you receive this error. You should add container instances to your cluster. For more information, see Launching an Amazon ECS Linux container instance.

Not enough ports

> If your task uses fixed host port mapping (for example, your task uses port 80 on the host for a web server), you must have at least one container instance per task, because only one container can use a single host port at a time. You should add container instances to your cluster or reduce your number of desired tasks.

Too many ports registered

> The closest matching container instance for task placement can't exceed the maximum allowed reserved port limit of 100 host ports per container instance. Using dynamic host port mapping may remediate the issue.

Port already in-use

The task definition of this task uses the same port in its port mapping as a task already running on the container instance that was chosen. The service event message would have the chosen container instance ID as part of the message below.

```
The closest matching container-instance is already using a port required by your
 task.
```

Not enough memory

If your task definition specifies 1000 MiB of memory, and the container instances in your cluster each have 1024 MiB of memory, you can only run one copy of this task per container instance. You can experiment with less memory in your task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

Note

If you're trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see Reserving Amazon ECS Linux container instance memory.

Not enough CPU

A container instance has 1,024 CPU units for every CPU core. If your task definition specifies 1,000 CPU units, and the container instances in your cluster each have 1,024 CPU units, you can only run one copy of this task per container instance. You can experiment with fewer CPU units in your task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

Not enough available ENI attachment points

Tasks that use the `awsvpc` network mode each receive their own elastic network interface (ENI), which is attached to the container instance that hosts it. Amazon EC2 instances have a limit to the number of ENIs that can be attached to them and there are no container instances in the cluster that have ENI capacity available.

The ENI limit for individual container instances depends on the following conditions:

- If you **have not** opted in to the `awsvpcTrunking` account setting, the ENI limit for each container instance depends on the instance type. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide*.
- If you **have** opted in to the `awsvpcTrunking` account setting but you **have not** launched new container instances using a supported instance type after opting in, the ENI limit for each container instance is still at the default value. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide*.
- If you **have** opted in to the `awsvpcTrunking` account setting and you **have** launched new container instances using a supported instance type after opting in, additional ENIs are available. For more information, see Supported instances for increased Amazon ECS container network interfaces.

For more information about opting in to the `awsvpcTrunking` account setting, see Increasing Amazon ECS Linux container instance network interfaces.

You can add container instances to your cluster to provide more available network adapters.

Container instance missing required attribute

Some task definition parameters require a specific Docker remote API version to be installed on the container instance. Others, such as the logging driver options, require the container instances to register those log drivers with the `ECS_AVAILABLE_LOGGING_DRIVERS` agent configuration variable. If your task definition contains a parameter that requires a specific container instance attribute, and you don't have any available container instances that can satisfy this requirement, the task can't be placed.

A common cause of this error is if your service is using tasks that use the `awsvpc` network mode and the EC2 launch type. The cluster you specified doesn't have a container instance registered to it in the same subnet that was specified in the `awsvpcConfiguration` when the service was created.

You can use the AWSSupport-TroubleshootECSContainerInstance runbook to troubleshoot. The runbook reviews whether the user data for the instance contains the correct cluster information, whether the instance profile contains the required permissions, and network configuration issues. For more information, see AWSSupport-TroubleshootECSContainerInstance in the *AWS Systems Manager Automation runbook reference User Guide*.

For more information on which attributes are required for specific task definition parameters and agent configuration variables, see Amazon ECS task definition parameters for the Fargate launch type and Amazon ECS container agent configuration.

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* has insufficient CPU units available.

The closest matching container instance for task placement doesn't contain enough CPU units to meet the requirements in the task definition. Review the CPU requirements in both the task size and container definition parameters of the task definition.

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* encountered error "AGENT".

The Amazon ECS container agent on the closest matching container instance for task placement is disconnected. If you can connect to the container instance with SSH, you can examine the agent logs; for more information, see Amazon ECS container agent log configuration parameters. You should also verify that the agent is running on the instance. If you are using the Amazon ECS-optimized AMI, you can try stopping and restarting the agent with the following command.

- For the Amazon ECS-optimized Amazon Linux 2 AMI and Amazon ECS-optimized Amazon Linux 2023 AMI

  ```
  sudo systemctl restart ecs
  ```

- For the Amazon ECS-optimized Amazon Linux AMI

  ```
  sudo stop ecs && sudo start ecs
  ```

service (*service-name*) (task *task-id*) (instance *instance-id*) is unhealthy in (elb *elb-name*) due to (reason Instance has failed at least the UnhealthyThreshold number of health checks consecutively.)

This service is registered with a load balancer and the load balancer health checks are failing. The message includes the task ID to help identify which specific task is failing health checks. For more information, see Troubleshooting service load balancers in Amazon ECS.

service (*service-name*) is unable to consistently start tasks successfully.

This service contains tasks that have failed to start after consecutive attempts. At this point, the service scheduler begins to incrementally increase the time between retries. You should troubleshoot why your tasks are failing to launch. For more information, see Amazon ECS service throttle logic.

After the service is updated, for example with an updated task definition, the service scheduler resumes normal behavior.

service (*service-name*) operations are being throttled. Will try again later.

This service is unable to launch more tasks due to API throttling limits. Once the service scheduler is able to launch more tasks, it will resume.

To request an API rate limit quota increase, open the AWS Support Center page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.

service (*service-name*) was unable to stop or start tasks during a deployment because of the service deployment configuration. Update the minimumHealthyPercent or maximumPercent value and try again.

This service is unable to stop or start tasks during a service deployment due to the deployment configuration. The deployment configuration consists of the `minimumHealthyPercent` and `maximumPercent` values, which are defined when the service is created. Those values can also be updated on an existing service.

The `minimumHealthyPercent` represents the lower limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining. It's a percent of the desired number of tasks for the service. This value is rounded up. For example, if the minimum healthy percent is `50` and the desired task count is four, then the scheduler can stop two existing tasks before starting two new tasks. Likewise, if the minimum healthy percent is 75% and the desired task count is two, then the scheduler can't stop any tasks due to the resulting value also being two.

The `maximumPercent` represents the upper limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining. It's a percent of the desired number of tasks for a service. This value is rounded down. For example, if the maximum percent is `200` and the desired task count is four, then the scheduler can start four new tasks before stopping four existing tasks. Likewise, if the maximum percent is `125` and the desired task count is three, the scheduler can't start any tasks due to the resulting value also being three.

When setting a minimum healthy percent or a maximum percent, you should ensure that the scheduler can stop or start at least one task when a deployment is triggered.

service (*service-name*) was unable to place a task. Reason: You've reached the limit on the number of tasks you can run concurrently

You can request a quota increase for the resource that caused the error. For more information, see *Service quotas*. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

service (*service-name*) was unable to place a task. Reason: Internal error.

The following is the possible reason for this error:

The service is unable to start a task due to a subnet being in an unsupported Availability Zone.

For information about the supported Fargate Regions and Availability Zones, see the section called “AWS Fargate Regions”.

For information about how to view the subnet Availability Zone, see View your subnet in the *Amazon VPC User Guide*.

service (*service-name*) was unable to place a task. Reason: The requested CPU configuration is above your limit.

You can request a quota increase for the resource that caused the error. For more information, see *Service quotas*. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

service (*service-name*) was unable to place a task. Reason: The requested MEMORY configuration is above your limit.

You can request a quota increase for the resource that caused the error. For more information, see *Service quotas*. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

service (*service-name*) was unable to place a task. Reason: You've reached the limit on the number of vCPUs you can run concurrently

AWS Fargate is transitioning from task count-based quotas to vCPU-based quotas.

You can request a quota increase for the Fargate vCPU-based quota. For more information, see *Service quotas*. To request a Fargate quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

service (*service-name*) was unable to reach steady state because task set (*taskSet-ID*) was unable to scale in. Reason: The number of protected tasks are more than the desired count of tasks

The service has more protected tasks than the desirednumber of tasks. You can do one the following:

- Wait until the protection on the current tasks expire, enabling them to be terminated.
- Determine which tasks can be stopped and use the `UpdateTaskProtection` API with the `protectionEnabled` option set to `false` to unset protection for these tasks.
- Increase the desired task count of the service to more than the number of protected tasks.

service (*service-name*) was unable to reach steady state. Reason: No Container Instances were found in your capacity provider.

The service scheduler sends this event message when it couldn't find the available resources to add another task. The possible causes for this are:

There is no capacity provider associated with the cluster

> Use `describe-services` to verify that you have a capacity provider associated with the cluster You can update the capacity provider strategy for the service.

Verify that there is available capacity in the capacity provider, In the case of the EC2 launch type, make sure that the container instances meet the task definition requirements.

No container instances were found in your cluster

If no container instances are registered in the cluster you attempt to run a task in, you receive this error. You should add container instances to your cluster. For more information, see Launching an Amazon ECS Linux container instance.

Not enough ports

If your task uses fixed host port mapping (for example, your task uses port 80 on the host for a web server), you must have at least one container instance per task. Only one container can use a single host port at a time. You should add container instances to your cluster or reduce your number of desired tasks.

Too many ports registered

The closest matching container instance for task placement can't exceed the maximum allowed reserved port limit of 100 host ports per container instance. Using dynamic host port mapping may remediate the issue.

Port already in-use

The task definition of this task uses the same port in its port mapping as a task already running on the container instance that was chosen. The service event message would have the chosen container instance ID as part of the message below.

```
The closest matching container-instance is already using a port required by your
 task.
```

Not enough memory

If your task definition specifies 1000 MiB of memory, and the container instances in your cluster each have 1024 MiB of memory, you can only run one copy of this task per container instance. You can experiment with less memory in your task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

> **Note**
>
> If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see Reserving Amazon ECS Linux container instance memory.

Not enough available ENI attachment points

Tasks that use the `awsvpc` network mode each receive their own elastic network interface (ENI), which is attached to the container instance that hosts it. Amazon EC2 instances have a limit to the number of ENIs that can be attached to them, and there are no container instances in the cluster that have ENI capacity available.

The ENI limit for individual container instances depends on the following conditions:

- If you **have not** opted in to the `awsvpcTrunking` account setting, the ENI limit for each container instance depends on the instance type. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide*.
- If you **have** opted in to the `awsvpcTrunking` account setting but you **have not** launched new container instances using a supported instance type after opting in, the ENI limit for each container instance is still at the default value. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide*.
- If you **have** opted in to the `awsvpcTrunking` account setting and you **have** launched new container instances using a supported instance type after opting in, additional ENIs are available. For more information, see Supported instances for increased Amazon ECS container network interfaces.

For more information about opting in to the `awsvpcTrunking` account setting, see Increasing Amazon ECS Linux container instance network interfaces.

You can add container instances to your cluster to provide more available network adapters.

Container instance missing required attribute

Some task definition parameters require a specific Docker remote API version to be installed on the container instance. Others, such as the logging driver options, require the container instances to register those log drivers with the `ECS_AVAILABLE_LOGGING_DRIVERS` agent configuration variable. If your task definition contains a parameter that requires a specific container instance attribute, and you don't have any available container instances that can satisfy this requirement, the task cannot be placed.

A common cause of this error is if your service is using tasks that use the `awsvpc` network mode and the EC2 launch type and the cluster you specified doesn't have a container instance registered to it in the same subnet that was specified in the `awsvpcConfiguration` when the service was created.

You can use the AWSSupport-TroubleshootECSContainerInstance runbook to troubleshoot. The runbook reviews whether the user data for the instance contains the correct cluster information, whether the instance profile contains the required permissions, and network configuration issues. For more information, see AWSSupport-TroubleshootECSContainerInstance in the *AWS Systems Manager Automation runbook reference User Guide*.

For more information on which attributes are required for specific task definition parameters and agent configuration variables, see Amazon ECS task definition parameters for the Fargate launch type and Amazon ECS container agent configuration.

service (*`service-name`*) was unable to place a task. Reason: Capacity is unavailable at this time. Please try again later or in a different availability zone.

There is currently no available capacity to run your service on.

You can do one the following:

- Wait until the Fargate capacity or EC2 container instances become available.
- Relaunch the service and specify additional subnets.

service (*`service-name`*) deployment failed: tasks failed to start.

The tasks in your service failed to start.

For information about how to debug stopped tasks. see Amazon ECS stopped tasks error messages.

service (*`service-name`*) Timed out waiting for Amazon ECS Agent to start. Please check logs at /var/log/ecs/ecs-agent.log".

The Amazon ECS container agent on the closest matching container instance for task placement is disconnected. If you can connect to the container instance with SSH, you can examine the agent logs. For more information, see Amazon ECS container agent log configuration parameters. You should also verify that the agent is running on the instance. If you are using the Amazon ECS-optimized AMI, you can try stopping and restarting the agent with the following command.

- For the Amazon ECS-optimized Amazon Linux 2 AMI

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI

```
sudo stop ecs && sudo start ecs
```

service (*`service-name`*) task set (*`taskSet-ID`*) (task *`task-id`*) is not healthy in target-group (*`targetGroup-ARN)`*) due to `TARGET GROUP IS NOT FOUND`.

The task set for the service is failing health checks because the target group isn't found. The message includes the task ID to help identify which specific task is failing health checks. You should delete and recreate the service. Don't delete any Elastic Load Balancing target group unless the corresponding Amazon ECS service is already deleted.

service (*`service-name`*) task set (*`taskSet-ID`*) (task *`task-id`*) is not healthy in target-group (*`targetGroup-ARN)`*) due to `TARGET IS NOT FOUND`.

The task set for the service is failing health checks because the target isn't found. The message includes the task ID to help identify which specific task is failing health checks.

IAM permissions policies have been misconfigured or changed, and ECS can no longer maintain your service

The service is unable to maintain tasks due to misconfigured or changed IAM permissions policies. The IAM role associated with your ECS service or tasks may be missing required permissions.

To resolve this issue, add the necessary permissions to the IAM role. For more information about managing IAM permissions policies, see Adding and removing IAM identity permissions in the *IAM User Guide*.

IAM trust relationship has been misconfigured or changed, and ECS can no longer maintain your service

The service is unable to maintain tasks due to a misconfigured or changed IAM trust relationship. The IAM role associated with your ECS service or tasks may have an incorrect trust policy.

To resolve this issue, configure a trust policy for the role used in your task definition. For more information about creating trust policies for custom roles, see Creating a role for a custom use case in the *IAM User Guide*.

Amazon ECS service unhealthy event messages

The following are examples of service unhealthy event messages.

EC2: (service *service-name*) (task *task-id*) (instance *instance-id*) (port *port-number*) is unhealthy in (target-group *target-group-name*) due to (reason *failure-reason*)

This message indicates that a task running on an EC2 instance is failing health checks. For more information, see the following:

- How do I get my Amazon ECS tasks that use the Amazon EC2 launch type to pass the Application Load Balancer health check?

EC2 with taskSet: (service *service-name*, taskSet *taskSet-id*) (task *task-id*) (instance *instance-id*) (port *port-number*) is unhealthy in (target-group *target-group-name*) due to (reason *failure-reason*)

This message indicates that a task in a task set running on an EC2 instance is failing health checks. For more information, see the following:

- How do I get my Amazon ECS tasks that use the Amazon EC2 launch type to pass the Application Load Balancer health check?

Fargate: (service *service-name*) (task *task-id*) (port *port-number*) is unhealthy in (target-group *target-group-name*) due to (reason *failure-reason*)

This message indicates that a Fargate task is failing health checks.

For more information about troubleshooting health check failures in Fargate tasks, see How do I troubleshoot health check failures for Amazon ECS tasks on Fargate?.

Fargate with taskSet: (service *service-name*, taskSet *taskSet-id*) (task *task-id*) (port *port-number*) is unhealthy in (target-group *target-group-name*) due to (reason *failure-reason*)

This message indicates that a task in a task set running on Fargate is failing health checks.

For more information about troubleshooting health check failures in Fargate tasks, see How do I troubleshoot health check failures for Amazon ECS tasks on Fargate?.

Amazon ECS Availability Zone service rebalancing service event messages

The following are examples of service event messages you might see.

service (*service-name*) is not AZ balanced with *number-tasks* tasks in *Availability Zone 1*, *number-tasks* in *Availability Zone 2*, and *number-tasks* in *Availability Zone 3*. AZ Rebalancing in progress.

The service scheduler sends a `service (service-name) is not AZ balanced` service event when the number of tasks is not evenly spread across Availability Zones. There is no action to take. This is an information event.

service (*service-name*) is AZ balanced with *number-tasks* tasks in *Availability Zone 1*, *number-tasks* tasks in *Availability Zone 2*, and *number-tasks* tasks in *Availability Zone 3*.

The service scheduler sends a `service (service-name) is AZ balanced` service event when Availability Zone service rebalancing completes. There is no action to take. This is an information event.

service-name has started *number-tasks* tasks in *Availability Zone* to AZ Rebalance: *task-ids*.

The service scheduler sends a *service-name*/*task-set-name* has started *number* tasks in *Availability Zone* service event when it starts tasks in an Availability Zone because of service rebalancing. There is no action to take. This is an information event.

service-name has stopped *number-tasks* running tasks in *Availability Zone* due to AZ rebalancing: *task-id*.

The service scheduler sends a *service-name*/*task-set-name* has stopped *number* tasks in *Availability Zone* service event when it stops tasks in an Availability Zone because of service rebalancing. There is no action to take. This is an information event.

service (*service-name*) is unable to place a task in *Availability Zone* because no container instance met all of its requirements.

The service scheduler sends a *service-name* is unable to place a task in *Availability Zone* service event because no container instance met all of its requirements. To address the issue, launch instances in the Availability Zone.

service (*service-name*) is unable to place a task in *Availability Zone*.

The service scheduler sends a *service-name* is unable to place a task in *Availability Zone* service event when you use the Fargate launch type and there is no available capacity.

You can add additional subnets in the Availability Zone in the error message, or contact Support to get additional capacity.

service (*service-name*) was unable to AZ Rebalance because *task-set-name* was unable to scale in due to *reason*.

The service scheduler sends a *service-name* was unable to AZ Rebalance because *task-set-name* was unable to scale in due to *reason* service event when you use task scale-in protection.

You can do one the following:

- Wait until the protection on the current tasks expire, enabling them to be terminated.
- Determine which tasks can be stopped and use the `UpdateTaskProtection` API with the `protectionEnabled` option set to `false` to unset protection for these tasks.

- Increase the desired task count of the service to more than the number of protected tasks.

service (*service-name*) stopped AZ Rebalancing.

The service scheduler sends a *service-name* stopped AZ Rebalancing service event when the Availability Zone rebalancing operation stopped. This is an information event. Amazon ECS sends additional events which provide more information.

Troubleshooting service load balancers in Amazon ECS

Amazon ECS services can register tasks with an Elastic Load Balancing load balancer. Load balancer configuration errors are common causes for stopped tasks. If your stopped tasks were started by services that use a load balancer, consider the following possible causes.

Amazon ECS service-linked role doesn't exist

The Amazon ECS service-linked role allows Amazon ECS services to register container instances with Elastic Load Balancing load balancers. The service-linked role must be created in your account. For more information, see Using service-linked roles for Amazon ECS.

Container instance security group

If your container is mapped to port 80 on your container instance, your container instance security group must allow inbound traffic on port 80 for the load balancer health checks to pass.

Elastic Load Balancing load balancer not configured for all Availability Zones

Your load balancer should be configured to use all of the Availability Zones in a Region, or at least all of the Availability Zones where your container instances reside. If a service uses a load balancer and starts a task on a container instance that resides in an Availability Zone that the load balancer isn't configured to use, the task never passes the health check. This results in the task being killed.

Elastic Load Balancing load balancer health check misconfigured

The load balancer health check parameters can be overly restrictive or point to resources that don't exist. If a container instance is determined to be unhealthy, it's removed from the load balancer. Be sure to verify that the following parameters are configured correctly for your service load balancer.

Ping Port

The **Ping Port** value for a load balancer health check is the port on the container instances that the load balancer checks to determine if it is healthy. If this port is misconfigured, the load balancer likely deregisters your container instance from itself. This port should be configured to use the `hostPort` value for the container in your service's task definition that you're using with the health check.

Ping Path

This is part of the load balancer healthcheck. It is an endpoint on your application that can resturn a successful status code (for example, 200) when the application is healthy. This value is often set to `index.html`, but if your service doesn't respond to that request, then the health check fails. If your container doesn't have an `index.html` file, you can set this to / to target the base URL for the container instance.

Response Timeout

This is the amount of time that your container has to return a response to the health check ping. If this value is lower than the amount of time required for a response, the health check fails.

Health Check Interval

This is the amount of time between health check pings. The shorter your health check intervals are, the faster your container instance can reach the **Unhealthy Threshold**.

Unhealthy Threshold

This is the number of times your health check can fail before your container instance is considered unhealthy. If you have an unhealthy threshold of 2, and a health check interval of 30 seconds, then your task has 60 seconds to respond to the health check ping before it's assumed unhealthy. You can raise the unhealthy threshold or the health check interval to give your tasks more time to respond.

Unable to update the service *`servicename`*: Load balancer container name or port changed in task definition

If your service uses a load balancer, you can use the AWS CLI or SDK to modify the load balancer configuration. For information about how to modify the configuration, see UpdateService in the *Amazon Elastic Container Service API Reference*. If you update the task definition for the service, the container name and container port that are specified in the load balancer configuration must remain in the task definition.

You've reached the limit on the number of tasks that you can run concurrently.

For a new account, your quotas might be lower that the service quotas. The service quota for your account can be viewed in the Service Quotas console. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

Troubleshooting service auto scaling in Amazon ECS

Application Auto Scaling turns off scale-in processes while Amazon ECS deployments are in progress, and they resume once the deployment has completed. However, scale-out processes continue to occur, unless suspended, during a deployment. For more information, see Suspending and resuming scaling for Application Auto Scaling.

Troubleshoot Amazon ECS task definition invalid CPU or memory errors

When registering a task definition using the Amazon ECS API or AWS CLI, if you specify an invalid `cpu` or `memory` value, the following error is returned.

```
An error occurred (ClientException) when calling the RegisterTaskDefinition operation:
 Invalid 'cpu' setting for task.
```

Note

When using Terraform, the following error might be returned.

```
Error: ClientException: No Fargate configuration exists for given values.
```

To resolve this issue, you must specify a supported value for the task CPU and memory in your task definition. The `cpu` value can be expressed in CPU units or vCPUs in a task definition. It's converted to an integer indicating the CPU units when the task definition is registered. The `memory` value can be expressed in MiB or GB in a task definition. It's converted to an integer indicating the MiB when the task definition is registered.

For task definitions that specify FARGATE for the requiresCompatibilities parameter (even if EC2 is also specified), you must use one of the values in the following table. These values determine your range of supported values for the CPU and memory parameter.

For tasks hosted on Fargate, the following table shows the valid CPU and memory combinations. The memory values in the JSON file are specified in MiB. You can convert the GB value to MiB by multiplying the value by 1024. For example 1 GB = 1024 MiB.

CPU value	Memory value	Operating systems supported for AWS Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows
8192 (8 vCPU) **Note** This option requires Linux platform `1.4.0` or later.	Between 16 GB and 60 GB in 4 GB increments	Linux
16384 (16vCPU)	Between 32 GB and 120 GB in 8 GB increments	Linux

CPU value	Memory value	Operating systems supported for AWS Fargate
Note This option requires Linux platform `1.4.0` or later.		

For tasks hosted on Amazon EC2, supported task CPU values are between 0.25 vCPUs and 192 vCPUs.

The CPU control mechanism differs between Amazon EC2 and Fargate launch types:

- For tasks hosted on Amazon EC2: Amazon ECS uses the CPU period and the CPU quota to control the task size CPU hard limits. When you specify the vCPU in your task definition, Amazon ECS translates the value to the CPU period and CPU quota settings that apply to the `cgroup`.
- For tasks hosted on Fargate: Amazon ECS uses CPU shares to control CPU allocation. The CPU quota and period values are not used for CPU limiting in Fargate tasks.

For Amazon EC2 tasks, the CPU quota controls the amount of CPU time granted to a `cgroup` during a given CPU period. Both settings are expressed in terms of microseconds. When the CPU quota equals the CPU period means a `cgroup` can execute up to 100% on one vCPU (or any other fraction that totals to 100% for multiple vCPUs). The CPU quota has a maximum of 1000000us and the CPU period has a minimum of 1ms. You can use these values to set the limits for your CPU count. When you change the CPU period without changing the CPU quota, you have different effective limits than what you've specified in your task definition.

The 100ms period allows for vCPUs ranging from 0.125 to 10.

> **Note**
>
> Task-level CPU and memory parameters are ignored for Windows containers.

Viewing Amazon ECS container agent logs

Amazon ECS stores logs in the `/var/log/ecs` folder of your container instances. There are logs available from the Amazon ECS container agent and from the `ecs-init` service that controls the state of the agent (start/stop) on the container instance. You can view these log files by connecting to a container instance using SSH.

> **Note**
>
> If you are not sure how to collect all of the logs on your container instances, you can use the Amazon ECS logs collector. For more information, see Collecting container logs with Amazon ECS logs collector .

Linux operating system

The `ecs-init` process stores logs at `/var/log/ecs/ecs-init.log`.

The `ecs-init.log` file contains information about the container agent lifecycle management, configuration, and bootstrapping.

You can use the following command to view the log files.

```
cat /var/log/ecs/ecs-init.log
```

Output:

```
2018-02-16T18:13:54Z [INFO] pre-start
2018-02-16T18:13:56Z [INFO] start
2018-02-16T18:13:56Z [INFO] No existing agent container to remove.
2018-02-16T18:13:56Z [INFO] Starting Amazon Elastic Container Service Agent
```

Windows operating system

You can use the Amazon ECS logs collector for Windows. For more information, see Amazon ECS Logs Collector For Windows on Github.

1. Connect to your instance.
2. Open PowerShell and then run the following commands with administrative privileges. The commands download the script and collects the logs.

```
Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://
raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-
logs-collector.ps1
.\ecs-logs-collector.ps1
```

You can turn on debug logging for Amazon ECS agent and the Docker daemon. This option allows the script to collect the logs before turning on debug mode. The script restarts the Docker daemon and Amazon ECS agent, and then terminates all containers running on the instance. Before running the following command, drain the container instance and moving any important tasks to other container instances.

Run the following command to turn on logging.

```
.\ecs-logs-collector.ps1 -RunMode debug
```

Collecting container logs with Amazon ECS logs collector

If you are unsure how to collect all of the various logs on your container instances, you can use the Amazon ECS logs collector. It is available on GitHub for both Linux and Windows. The script collects general operating system logs as well as Docker and Amazon ECS container agent logs, which can be helpful for troubleshooting AWS Support cases. It then compresses and archives the collected information into a single file that can easily be shared for diagnostic purposes. It also supports enabling debug mode for the Docker daemon and the Amazon ECS container agent on Amazon Linux variants, such as the Amazon ECS-optimized AMI. Currently, the Amazon ECS logs collector supports the following operating systems:

- Amazon Linux
- Red Hat Enterprise Linux 7
- Debian 8
- Ubuntu 14.04
- Ubuntu 16.04
- Ubuntu 18.04
- Windows Server 2016

> **Note**
>
> The source code for the Amazon ECS logs collector is available on GitHub for both Linux and Windows. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services doesn't currently support running modified copies of this software.

To download and run the Amazon ECS logs collector for Linux

1. Connect to your container instance.
2. Download the Amazon ECS logs collector script.

```
curl -O https://raw.githubusercontent.com/awslabs/ecs-logs-collector/master/ecs-logs-collector.sh
```

3. Run the script to collect the logs and create the archive.

> **Note**
>
> To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `--mode=enable-debug` option to the following command. This might restart the Docker daemon, which kills all containers that are running on the instance. Consider draining the container instance and moving any important tasks to other container instances before enabling debug mode. For more information, see Draining Amazon ECS container instances.

```
[ec2-user ~]$ sudo bash ./ecs-logs-collector.sh
```

> **Important**
>
> We recommend that you edit the logs and remove all sensitive data from the files. You can search for known data, and also search for environment variables such as `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` in the file.

After you have run the script, you can examine the collected logs in the `collect` folder that the script created. The `collect.tgz` file is a compressed archive of all of the logs, which you can share with AWS Support for diagnostic help.

To download and run the Amazon ECS logs collector for Windows

1. Connect to your container instance. For more information, see [Connect to your Windows instance using RDP]() in the *Amazon EC2 User Guide*.
2. Download the Amazon ECS logs collector script using PowerShell.

   ```
   Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://
   raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-
   logs-collector.ps1
   ```

3. Run the script to collect the logs and create the archive.

   > **Note**
   >
   > To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `-RunMode debug` option to the following command. This restarts the Docker daemon, which kills all containers that are running on the instance. Consider draining the container instance and moving any important tasks to other container instances before enabling debug mode. For more information, see [Draining Amazon ECS container instances]().

   ```
   .\ecs-logs-collector.ps1
   ```

> **Important**
>
> We recommend that you edit the logs and remove all sensitive data from the files. You can search for known data, and also search for environment variables such as `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` in the file.

After you have run the script, you can examine the collected logs in the `collect` folder that the script created. The `collect.tgz` file is a compressed archive of all of the logs, which you can share with AWS Support for diagnostic help.

Retrieve Amazon ECS diagnostic details with agent introspection

The Amazon ECS agent introspection API provides information about the overall state of the Amazon ECS agent and the container instances.

You can use the agent introspection API to get the Docker ID for a container in your task. You can use the agent introspection API by connecting to a container instance using SSH.

> **Important**
>
> Your container instance must have an IAM role that allows access to Amazon ECS in order to reach the introspection API. For more information, see Amazon ECS container instance IAM role.

The following example shows two tasks, one that is currently running and one that was stopped.

> **Note**
>
> The following command is piped through the **python -mjson.tool** for greater readability.

```
curl http://localhost:51678/v1/tasks | python -mjson.tool
```

Output:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1095  100  1095    0     0   117k      0 --:--:-- --:--:-- --:--:--  133k
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/090eff9b-1ce3-4db6-848a-
a8d14064fd24",
```

```
            "Containers": [
                {
                    "DockerId":
 "189a8ff4b5f04affe40e5160a5ffadca395136eb5faf4950c57963c06f82c76d",
                    "DockerName": "ecs-console-sample-app-static-6-simple-
app-86caf9bcabe3e9c61600",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "f7f1f8a7a245c5da83aa92729bd28c6bcb004d1f6a35409e4207e1d34030e966",
                    "DockerName": "ecs-console-sample-app-static-6-busybox-
ce83ce978a87a890ab01",
                    "Name": "busybox"
                }
            ],
            "Family": "console-sample-app-static",
            "KnownStatus": "STOPPED",
            "Version": "6"
        },
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/1810e302-eaea-4da9-
a638-097bea534740",
            "Containers": [
                {
                    "DockerId":
 "dc7240fe892ab233dbbcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
                    "DockerName": "ecs-console-sample-app-static-6-simple-app-
f0e5859699a7aecfb101",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "096d685fb85a1ff3e021c8254672ab8497e3c13986b9cf005cbae9460b7b901e",
                    "DockerName": "ecs-console-sample-app-static-6-
busybox-92e4b8d0ecd0cce69a01",
                    "Name": "busybox"
                }
            ],
            "DesiredStatus": "RUNNING",
            "Family": "console-sample-app-static",
            "KnownStatus": "RUNNING",
            "Version": "6"
        }
```

```
    ]
}
```

In the preceding example, the stopped task (*090eff9b-1ce3-4db6-848a-a8d14064fd24*) has two containers. You can use **docker inspect *container-ID*** to view detailed information on each container. For more information, see Amazon ECS container introspection.

Docker diagnostics in Amazon ECS

Docker provides several diagnostic tools that help you troubleshoot problems with your containers and tasks. For more information about all of the available Docker command line utilities, see the Docker CLI reference in the Docker documentation. You can access the Docker command line utilities by connecting to a container instance using SSH.

The exit codes that Docker containers report can also provide some diagnostic information (for example, exit code 137 means that the container received a `SIGKILL` signal). For more information, see Exit Status in the Docker documentation.

List Docker containers in Amazon ECS

You can use the **docker ps** command on your container instance to list the running containers. In the following example, only the Amazon ECS container agent is running. For more information, see docker ps in the Docker documentation.

```
docker ps
```

Output:

```
CONTAINER ID        IMAGE                            COMMAND             CREATED
      STATUS              PORTS                        NAMES
cee0d6986de0        amazon/amazon-ecs-agent:latest   "/agent"            22 hours ago
      Up 22 hours         127.0.0.1:51678->51678/tcp   ecs-agent
```

You can use the **docker ps -a** command to see all containers (even stopped or killed containers). This is helpful for listing containers that are unexpectedly stopping. In the following example, container `f7f1f8a7a245` exited 9 seconds ago, so it doesn't show up in a **docker ps** output without the `-a` flag.

```
docker ps -a
```

Output:

```
CONTAINER ID        IMAGE                                            COMMAND
 CREATED             STATUS                         PORTS                     NAMES
db4d48e411b1        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"
  19 seconds ago                                                                ecs-
console-sample-app-static-6-internalecs-emptyvolume-source-c09288a6b0cba8a53700
f7f1f8a7a245        busybox:buildroot-2014.02                   "\"sh -c '/bin/sh -c
  22 hours ago        Exited (137) 9 seconds ago                                ecs-
console-sample-app-static-6-busybox-ce83ce978a87a890ab01
189a8ff4b5f0        httpd:2                                     "httpd-foreground"
  22 hours ago        Exited (137) 40 seconds ago                               ecs-
console-sample-app-static-6-simple-app-86caf9bcabe3e9c61600
0c7dca9321e3        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"
  22 hours ago                                                                  ecs-
console-sample-app-static-6-internalecs-emptyvolume-source-90fefaa68498a8a80700
cee0d6986de0        amazon/amazon-ecs-agent:latest              "/agent"
  22 hours ago        Up 22 hours                   127.0.0.1:51678->51678/tcp   ecs-
agent
```

View Docker Logs in Amazon ECS

You can view the STDOUT and STDERR streams for a container with the **docker logs** command. In this example, the logs are displayed for the *dc7240fe892a* container and piped through the **head** command for brevity. For more information, go to docker logs in the Docker documentation.

> **Note**
>
> Docker logs are only available on the container instance if you are using the default `json` log driver. If you have configured your tasks to use the `awslogs` log driver, then your container logs are available in CloudWatch Logs. For more information, see Send Amazon ECS logs to CloudWatch .

```
docker logs dc7240fe892a | head
```

Output:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
```

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
[Thu Apr 23 19:48:36.956682 2015] [mpm_event:notice] [pid 1:tid 140327115417472]
 AH00489: Apache/2.4.12 (Unix) configured -- resuming normal operations
[Thu Apr 23 19:48:36.956827 2015] [core:notice] [pid 1:tid 140327115417472] AH00094:
 Command line: 'httpd -D FOREGROUND'
10.0.1.86 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:29 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.0.154 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.1.86 - - [23/Apr/2015:19:49:58 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:50:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:50:29 +0000] "GET / HTTP/1.1" 200 348
time="2015-04-23T20:11:20Z" level="fatal" msg="write /dev/stdout: broken pipe"
```

Inspect Docker Containers in Amazon ECS

If you have the Docker ID of a container, you can inspect it with the **docker inspect** command. Inspecting containers provides the most detailed view of the environment in which a container was launched. For more information, see docker inspect in the Docker documentation.

```
docker inspect dc7240fe892a
```

Output:

```
[{
    "AppArmorProfile": "",
    "Args": [],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "httpd-foreground"
        ],
        "CpuShares": 10,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
```

```
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
local/apache2/bin",
            "HTTPD_PREFIX=/usr/local/apache2",
            "HTTPD_VERSION=2.4.12",
            "HTTPD_BZ2_URL=https://www.apache.org/dist/httpd/httpd-2.4.12.tar.bz2"
        ],
        "ExposedPorts": {
            "80/tcp": {}
        },
        "Hostname": "dc7240fe892a",
...
```

Configuring verbose output from the Docker daemon in Amazon ECS

If you're having trouble with Docker containers or images, you can turn on debug mode on your Docker daemon. Using debugging provides more verbose output from the daemon, You can use this to retrive error messages that are sent from container registries, such as Amazon ECR.

> **Important**
>
> This procedure is written for the Amazon ECS-optimized Amazon Linux AMI. For other operating systems, see Enable debugging and Control and configure Docker with systemd in the Docker documentation.

To use Docker daemon debug mode on the Amazon ECS-optimized Amazon Linux AMI

1. Connect to your container instance.
2. Open the Docker options file with a text editor, such as **vi**. For the Amazon ECS-optimized Amazon Linux AMI, the Docker options file is at `/etc/sysconfig/docker`.
3. Find the Docker options statement and add the `-D` option to the string, inside the quotes.

   > **Note**
   >
   > If the Docker options statement begins with a #, remove that character to uncomment the statement and enable the options.

For the Amazon ECS-optimized Amazon Linux AMI, the Docker options statement is called `OPTIONS`. For example:

```
# Additional startup options for the Docker daemon, for example:
# OPTIONS="--ip-forward=true --iptables=true"
# By default we limit the number of open files per container
OPTIONS="-D --default-ulimit nofile=1024:4096"
```

4. Save the file and exit your text editor.
5. Restart the Docker daemon.

```
sudo service docker restart
```

The output is as follows:

```
Stopping docker:                                          [  OK  ]
Starting docker: .                                  [  OK  ]
```

6. Restart the Amazon ECS agent.

```
sudo service ecs restart
```

Your Docker logs should now show more verbose output.

```
time="2015-12-30T21:48:21.907640838Z" level=debug msg="Unexpected response from
 server: \"{\\\"errors\\\":[{\\\"code\\\":\\\"DENIED\\\",\\\"message\\\":\\\"User:
 arn:aws:sts::1111:assumed-role/ecrReadOnly/i-abcdefg is not authorized to perform:
 ecr:InitiateLayerUpload on resource: arn:aws:ecr:us-east-1:1111:repository/nginx_test
\\\"}]}\\n\" http.Header{\"Connection\":[]string{\"keep-alive\"}, \"Content-Type\":
[]string{\"application/json; charset=utf-8\"}, \"Date\":[]string{\"Wed, 30 Dec 2015
 21:48:21 GMT\"}, \"Docker-Distribution-Api-Version\":[]string{\"registry/2.0\"},
 \"Content-Length\":[]string{\"235\"}}"
```

Troubleshoot the Docker API `error (500): devmapper` in Amazon ECS

The following Docker error indicates that the thin pool storage on your container instance is full, and that the Docker daemon cannot create new containers:

```
CannotCreateContainerError: API error (500): devmapper: Thin Pool has 4350 free data
 blocks which is less than minimum required 4454 free data blocks. Create more free
 space in thin pool or use dm.min_free_space option to change behavior
```

By default, Amazon ECS-optimized Amazon Linux AMIs from version `2015.09.d` and later launch with an 8-GiB volume for the operating system that's attached at `/dev/xvda` and mounted as the root of the file system. There's an additional 22-GiB volume that's attached at `/dev/xvdcz` that Docker uses for image and metadata storage. If this storage space is filled up, the Docker daemon cannot create new containers.

The easiest way to add storage to your container instances is to terminate the existing instances and launch new ones with larger data storage volumes. However, if you can't do this, you can add storage to the volume group that Docker uses and extend its logical volume by following the procedures in Amazon ECS-optimized Linux AMIs.

If your container instance storage is filling up too quickly, there are a few actions that you can take to reduce this effect:

- To view the thin poll information, run the following command on your container instance:

  ```
  docker info
  ```

- (Amazon ECS container agent 1.8.0 and later) You can reduce the amount of time that stopped or exited containers remain on your container instances. The `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` agent configuration variable sets the time duration to wait from when a task is stopped until the Docker container is removed (by default, this value is 3 hours). This removes the Docker container data. If this value is set too low, you might not be able to inspect your stopped containers or view the logs before they are removed. For more information, see Amazon ECS container agent configuration.
- You can remove non-running containers and unused images from your container instances. You can use the following example commands to manually remove stopped containers and unused

images. Deleted containers can't be inspected later, and deleted images must be pulled again before starting new containers from them.

To remove non-running containers, run the following command on your container instance:

```
docker rm $(docker ps -aq)
```

To remove unused images, run the following command on your container instance:

```
docker rmi $(docker images -q)
```

- You can remove unused data blocks within containers. You can use the following command to run **fstrim** on any running container and discard any data blocks that are unused by the container file system.

```
sudo sh -c "docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -
IZ fstrim /proc/Z/root/"
```

Troubleshoot Amazon ECS Exec issues

The following are troubleshooting notes to help diagnose why you may be getting an error when using ECS Exec.

Verify using the Exec Checker

The ECS Exec Checker script provides a way to verify and validate that your Amazon ECS cluster and task have met the prerequisites for using the ECS Exec feature. The ECS Exec Checker script verifies both your AWS CLI environment and cluster and tasks are ready for ECS Exec, by calling various APIs on your behalf. The tool requires the latest version of the AWS CLI and that the `jq` is available. For more information, see ECS Exec Checker on GitHub.

Error when calling `execute-command`

If a `The execute command failed` error occurs, the following are possible causes.

- The task does not have the required permissions. Verify that the task definition used to launch your task has a task IAM role defined and that the role has the required permissions. For more information, see ECS Exec permissions.

- The SSM agent isn't installed or isn't running.
- There is an interface Amazon VPC endpoint for Amazon ECS, but there isn't one for Systems Manager Session Manager.

Troubleshoot Amazon ECS Anywhere issues

Amazon ECS Anywhere provides support for registering an *external instance* such as an on-premises server or virtual machine (VM) to your Amazon ECS cluster. The following are common issues that you might encounter and general troubleshooting recommendations for them.

Topics

External instance registration issues

When registering an external instance with your Amazon ECS cluster, the following requirements must be met:

- An AWS Systems Manager activation, which consists of an *activation ID* and *activation code*, must be retrieved. You use it to register the external instance as a Systems Manager managed instance. When a Systems Manager activation is requested, specify a registration limit and expiration date. The registration limit specifies the maximum number of instances that can be registered using the activation. The default value for registration limit is 1 instance. The expiration date specifies when the activation expires. The default value is 24 hours. If the Systems Manager activation that you're using to register your external instance isn't valid, request a new one. For more information, see Registering an external instance to an Amazon ECS cluster.
- An IAM policy is used to provide your external instance the permissions that it needs to communicate with AWS API operations. If this managed policy isn't created properly and doesn't contain the required permissions, external instance registration fails. For more information, see Amazon ECS Anywhere IAM role.
- Amazon ECS provides an installation script that installs Docker, the Amazon ECS container agent, and the Systems Manager Agent on your external instance. If the installation script fails, it's

likely that the script can't be run again on the same instance without an error occurring. If this happens, follow the cleanup process to clear AWS resources from the instance so you can run the installation script again. For more information, see Deregistering an Amazon ECS external instance.

> **Note**
>
> Be aware that, if the installation script successfully requested and used the Systems Manager activation, running the installation script a second time uses the Systems Manager activation again. This might in turn cause you to reach the registration limit for that activation. If this limit is reached, you must create a new activation.

- When running the installation script on an external instance for GPU workloads, if the NVIDIA driver is not detected or configured properly, an error will occur. The installation script uses the `nvidia-smi` command to confirm the existence of the NVIDIA driver.

External instance network issues

To communicate any changes, your external instance requires a network connection to AWS. If your external instance loses its network connection to AWS, tasks that are running on your instances continue to run anyway unless stopped manually. After the connection to AWS is restored, the AWS credentials that are used by the Amazon ECS container agent and Systems Manager Agent on the external instance renew automatically. For more information about the AWS domains that are used for communication between your external instance and AWS, see Networking .

Issues running tasks on your external instance

If your tasks or containers fail to run on your external instance, the most common causes are either network or permission related. If your containers are pulling their images from Amazon ECR or are configured to send container logs to CloudWatch Logs, your task definition must specify a valid task execution IAM role. Without a valid task execution IAM role, your containers will fail to start. For more information about network related issues, see External instance network issues.

> **Important**
>
> Amazon ECS provides the Amazon ECS logs collection tool. You can use it to collect logs from your external instances for troubleshooting purposes. For more information, see [Collecting container logs with Amazon ECS logs collector](#).

Troubleshoot Java class loading issues on Fargate

Java applications running on Fargate may encounter class loading issues after platform updates, particularly when the application relies on non-deterministic class loading behavior. This can manifest as dependency injection errors, Spring Boot failures, or other runtime exceptions that were not present in previous deployments.

Symptoms

You might experience the following symptoms:

- Spring Boot dependency injection errors
- ClassNotFoundException or NoClassDefFoundError exceptions
- Applications that previously worked on Fargate now fail intermittently
- The same container image works on Amazon EC2 but fails on Fargate
- Inconsistent behavior across deployments with identical container images

Causes

These issues typically occur due to:

- **Non-deterministic class loading:** Java applications that depend on the order in which classes are loaded from JAR files may fail when the underlying platform changes how files are accessed or cached.
- **Platform updates:** Fargate platform version updates may change the underlying file system behavior, affecting the order in which classes are discovered and loaded.
- **JAR file ordering dependencies:** Applications that implicitly rely on specific JAR loading sequences without explicit dependency management.

Resolution

To resolve Java class loading issues on Fargate, implement deterministic class loading practices:

Immediate fix

If you need an immediate workaround:

1. **Enforce JAR loading order:** Explicitly specify the order in which JAR files should be loaded in your application's classpath configuration.
2. **Use explicit dependency management:** Ensure all dependencies are explicitly declared in your build configuration (Maven, Gradle, etc.) rather than relying on transitive dependencies.

Long-term best practices

Implement these practices to prevent future class loading issues:

1. **Make class loading deterministic:**
   - Use explicit dependency declarations in your build files
   - Avoid relying on classpath scanning order
   - Use dependency management tools to resolve version conflicts
   - Use the Java Virtual Machine (JVM) options such as `-verbose:class` to get information about classes loaded by JVM.
2. **Spring Boot applications:**
   - Use `@ComponentScan` with explicit base packages
   - Avoid auto-configuration conflicts by explicitly configuring beans
   - Use `@DependsOn` annotations to control bean initialization order
3. **Build configuration:**
   - Use dependency management sections in Maven or Gradle
   - Exclude transitive dependencies that cause conflicts
   - Use tools like Maven Enforcer Plugin to detect dependency issues
4. **Testing:**
   - Test your application with different JVM implementations
   - Run integration tests that simulate different deployment environments
   - Use tools to analyze classpath conflicts during development

Prevention

To prevent Java class loading issues in future deployments:

- **Follow deterministic class loading practices:** Design your application to not depend on the order in which classes are loaded from the classpath.
- **Use explicit dependency management:** Always explicitly declare all required dependencies and their versions in your build configuration.
- **Test across environments:** Regularly test your applications across different environments and platform versions to identify potential issues early.
- **Monitor platform updates:** Stay informed about Fargate platform updates and test your applications with new platform versions before they affect production workloads.

AWS Fargate throttling quotas

AWS Fargate limits Amazon ECS tasks and Amazon EKS pods launch rates to quotas (formerly referred to as limits) using a token bucket algorithm for each AWS account on a per-Region basis. With this algorithm, your account has a bucket that holds a specific number of tokens. The number of tokens in the bucket represents your rate quota at any given second. Each customer account has a tasks and pods token bucket that depletes based on the number of tasks and pods launched by the customer account. This token bucket has a bucket maximum that allows you to periodically make a higher number of requests, and a refill rate that allows you to sustain a steady rate of requests for as long as needed.

For example, the tasks and pods token bucket size for a Fargate customer account is 100 tokens, and the refill rate is 20 tokens per second. Therefore, you can immediately launch up to 100 Amazon ECS tasks and Amazon EKS pods per customer account, with a sustained launch rate of 20 Amazon ECS tasks and Amazon EKS pods per second.

Actions	Bucket maximum capacity (or Burst rate)	Bucket refill rate (or Sustained rate)
Fargate Resource rate quota for On Demand Amazon ECS tasks and Amazon EKS pods[1]	100	20

Actions	Bucket maximum capacity (or Burst rate)	Bucket refill rate (or Sustained rate)
Fargate Resource rate quota for Spot Amazon ECS tasks	100	20

[1]Accounts launching only Amazon EKS pods have a burst rate of 20 with a sustained pod launch rate of 20 pod launches per second when using the platform versions called out in the Amazon EKS platform versions.

Throttling the RunTask API in Fargate

In addition, Fargate limits the request rate when launching tasks using the Amazon ECS `RunTask` API using a separate quota. Fargate limits Amazon ECS `RunTask` API requests for each AWS account on a per-Region basis. Each request that you make removes one token from the bucket. We do this to help the performance of the service, and to ensure fair usage for all Fargate customers. API calls are subject to the request quotas whether they originate from the Amazon Elastic Container Service console, a command line tool, or a third-party application. The rate quota for calls to the Amazon ECS `RunTask` API is 20 calls per second (burst and sustained). Each call to this API can, however, launch up to 10 tasks. This means you can launch 100 tasks in one second by making 10 calls to this API, requesting 10 tasks to be launched in each call. Similarly, you could also make 20 calls to this API, requesting 5 tasks to be launched in each call. For more information on API throttling for Amazon ECS `RunTask` API, see API request throttling in the Amazon ECS API Reference.

In practice, task and pod launch rates are also dependent on other considerations such as container images to be downloaded and unpacked, health checks and other integrations enabled, such as registering tasks or pods into a load balancer. Customers see variations in task and pod launch rates compared with the quotas represented earlier based on the features that customers enable.

Adjusting rate quotas in Fargate

You can request an increase for Fargate rate throttling quotas for your AWS account. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

Amazon ECS API failure reasons

When an API action that you have triggered through the Amazon ECS API, console, or the AWS CLI exits with a `failures` error message, the following might assist in troubleshooting the cause. The failure returns a reason and the Amazon Resource Name (ARN) of the resource associated with the failure.

Many resources are Region-specific, so when using the console ensure that you set the correct Region for your resources. When using the AWS CLI, make sure that your AWS CLI commands are being sent to the correct Region with the `--region region` parameter.

For more information about the structure of the `Failure` data type, see Failure in the *Amazon Elastic Container Service API Reference*.

The following are examples of failure messages that you might receive when running API commands.

API action	Failure reason or Stopped reason	Cause
`DescribeClusters`	`MISSING`	The specified cluster wasn't found. Verify the spelling of the cluster name.
`DescribeInstances`	`MISSING`	The specified container instance wasn't found. Verify that you specified the cluster the container instance is registered to and that both the container instance ARN or ID is correct.
`DescribeServices`	`MISSING`	The specified service wasn't found. Verify that the correct cluster or Region is specified and that the service ARN or name is valid.

API action	Failure reason or Stopped reason	Cause
DescribeTasks	MISSING	The specified task wasn't found. Verify the correct cluster or Region is specified and that both the task ARN or ID is valid.

API action	Failure reason or Stopped reason	Cause
`DescribeTasks`	`TaskFailedToStart: RESOURCE:*`	For `RESOURCE:CPU` errors, the number of CPUs requested by the task are unavailable on your container instances. This generally happens when the CPU unit requirement in your task definition is larger than the CPU size of the Amazon EC2 instances defined in the Auto Scaling group mapped to the capacity provider. You need to check your capacity provider configuration. For `RESOURCE:MEMORY` errors, the amount of memory requested by the task are unavailable on your container instances. This generally happens when the memory amount requireme nt in your task definition is larger than the supported memory on the Amazon EC2 instances defined in the Auto Scaling group mapped to the capacity provider. You need to check your capacity provider configuration.

API action	Failure reason or Stopped reason	Cause
	`TaskFailedToStart: AGENT`	The container instance that you attempted to launch a task onto has an agent that's currently disconnected. To prevent extended wait times for task placement, the request was rejected. For information about how to troubleshoot an agent that's disconnected, see How do I troubleshoot a disconnected Amazon ECS agent.
	`TaskFailedToStart: MemberOf placement constraint unsatisfied`	There is no container instance that meets the placement constraints defined in your task definition.

API action	Failure reason or Stopped reason	Cause
	`TaskFailedToStart: ATTRIBUTE`	Your task definition contains a parameter that requires a specific container instance attribute that isn't available on your container instances. For example, if your task uses the `awsvpc` network mode, but there are no instances in your specified subnets with the `ecs.capability.task-eni` attribute. For more information about which attributes are required for specific task definition parameters and agent configuration variables, see [Amazon ECS task definition parameters for the Fargate launch type]() and [Amazon ECS container agent configuration]().
	`TaskFailedToStart: NO ACTIVE INSTANCES`	There are no active instances in your capacity provider. For information about how to manage your Auto Scaling groups, see [Auto Scaling groups]() in the *Amazon EC2 Auto Scaling User Guide*.

API action	Failure reason or Stopped reason	Cause
	`TaskFailedToStart: EMPTY CAPACITY PROVIDER`	There are no instances in your cluster. This is most likely because of an empty capacity provider, or because the instances in the capacity provider are not registered to the cluster. For information about how to manage your Auto Scaling groups, see [Auto Scaling groups](#) in the *Amazon EC2 Auto Scaling User Guide*.
`GetTaskProtection`	`MISSING`	The specified task wasn't found. Verify that the cluster name or ARN and the task ARN or ID are valid.
	`TASK_NOT_VALID`	The specified task isn't part of an Amazon ECS service. Only Amazon ECS service-managed tasks can be protected. Verify the task ARN or ID and try again.

API action	Failure reason or Stopped reason	Cause
`RunTask` or `StartTask`	`RESOURCE:*`	The resource or resources that are requested by the task are unavailable on the container instances in the cluster. If the resource is CPU, memory, ports, or elastic network interfaces, you might need to add additional container instances to your cluster. For `RESOURCE:ENI` errors, your cluster doesn't have any available elastic network interface attachment points, which are required for tasks that use the `awsvpc` network mode. Amazon EC2 instances have a limit to the number of network interfaces that can be attached to them, and the primary network interface counts as one. For more information about how many network interfaces are supported for each instance type, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide*. For `RESOURCE:GPU` errors, the number of GPUs requested by the task are unavailable and you might

API action	Failure reason or Stopped reason	Cause
		need to add GPU-enabled container instances to your cluster. For more information, see [Amazon ECS task definitions for GPU workloads]().
	`AGENT`	The container instance that you attempted to launch a task onto has an agent that's currently disconnected. To prevent extended wait times for task placement, the request was rejected. For information about how to troubleshoot an agent that's disconnected, see [How do I troubleshoot a disconnected Amazon ECS agent]().
	`LOCATION`	The container instance that you attempted to launch a task onto is in a different Availability Zone than the subnets that you specified in your `awsVpcConfiguration` .

API action	Failure reason or Stopped reason	Cause
	`ATTRIBUTE`	Your task definition contains a parameter that requires a specific container instance attribute that isn't available on your container instances. For example, if your task uses the `awsvpc` network mode, but there are no instances in your specified subnets with the `ecs.capability.task-eni` attribute. For more information about which attributes are required for specific task definition parameters and agent configuration variables, see [Amazon ECS task definition parameters for the Fargate launch type] and [Amazon ECS container agent configuration].
`StartTask`	`MISSING`	The container instance that you attempted to launch the task onto can't be found. Check if the wrong cluster or Region is specified, or the container instance ARN or ID is misspelled.

API action	Failure reason or Stopped reason	Cause
	INACTIVE	The container instance that you attempted to launch a task onto was previously deregistered with Amazon ECS and can't be used.
StopServiceDeployment	ECS deployment failed	A fraud account ran the StopServiceDeployment API.
TagResource	InvalidParameterException	The ARN for the service that you are tagging has the short format. You must migrate to the long format. For information about how to migrate the ARN, see [Migrate an Amazon ECS short service ARN to a long ARN]().
UpdateTaskProtection	DEPLOYMENT_BLOCKED	Can't set task protection as one or more protected tasks are preventing the service deployment from reaching a steady state. Unset task protection on existing tasks or wait until task protection expires.
	MISSING	The specified task wasn't found. Verify that the cluster name or ARN and the task ARN or ID are valid.

API action	Failure reason or Stopped reason	Cause
	`TASK_NOT_VALID`	The specified task isn't part of an Amazon ECS service. Only Amazon ECS service-managed tasks can be protected. Verify the task ARN or ID and try again.

Note

Besides the failure scenarios described here, API operations can also fail due to exceptions, resulting in error responses. For a list of such exceptions, see Common Errors.

Security in Amazon Elastic Container Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon Elastic Container Service, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon ECS. The following topics show you how to configure Amazon ECS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon ECS resources.

Topics

Identity and Access Management for Amazon Elastic Container Service

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon ECS resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon ECS.

Service user – If you use the Amazon ECS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon ECS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon ECS, see Troubleshooting Amazon Elastic Container Service identity and access.

Service administrator – If you're in charge of Amazon ECS resources at your company, you probably have full access to Amazon ECS. It's your job to determine which Amazon ECS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon ECS, see How Amazon Elastic Container Service works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon ECS. To view example Amazon ECS identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Elastic Container Service.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies]() in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choose between managed policies and inline policies]() in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal]() in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's

permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Elastic Container Service works with IAM

Before you use IAM to manage access to Amazon ECS, learn what IAM features are available to use with Amazon ECS.

IAM feature	Amazon ECS support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Partial
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Forward access sessions (FAS)	Yes
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how Amazon ECS and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon ECS

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon ECS

To view examples of Amazon ECS identity-based policies, see Identity-based policy examples for Amazon Elastic Container Service.

Resource-based policies within Amazon ECS

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon ECS

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon ECS actions, see Actions defined by Amazon Elastic Container Service in the *Service Authorization Reference*.

Policy actions in Amazon ECS use the following prefix before the action:

```
ecs
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "ecs:action1",
      "ecs:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "ecs:Describe*"
```

To view examples of Amazon ECS identity-based policies, see Identity-based policy examples for Amazon Elastic Container Service.

Policy resources for Amazon ECS

Supports policy resources: Partial

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon ECS resource types and their ARNs, see Resources defined by Amazon Elastic Container Service in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Elastic Container Service.

Some Amazon ECS API actions support multiple resources. For example, multiple clusters can be referenced when calling the `DescribeClusters` API action. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "EXAMPLE-RESOURCE-1",
      "EXAMPLE-RESOURCE-2"
```

For example, the Amazon ECS cluster resource has the following ARN:

```
arn:${Partition}:ecs:${Region}:${Account}:cluster/${clusterName}
```

To specify `my-cluster-1` and `my-cluster-2` cluster in your statement, use the following ARNs:

```
"Resource": [
         "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-1",
         "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-2"
```

To specify all clusters that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

For task definitions, you can specify the latest revision, or a specific revision.

To specify all revisions of the task definition, use the wildcard (*):

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}:*"
```

To specify a specific task definition revision, use ${TaskDefinitionRevisionNumber}:

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}:${TaskDefinitionRevisionNumber}"
```

To view examples of Amazon ECS identity-based policies, see Identity-based policy examples for Amazon Elastic Container Service.

Policy condition keys for Amazon ECS

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Amazon ECS supports the following service-specific condition keys that you can use to provide fine-grained filtering for your IAM policies:

Condition Key	Description	Evaluation Types
aws:RequestTag/${TagKey}	The context key is formatted `"aws:RequestTag/ tag-key":"tag-value "` where *tag-key* and *tag-value* are a tag key and value pair. Checks that the tag key–value pair is present in an AWS request. For example, you could check to see that the request includes the tag key `"Dept"` and that it has the value `"Accounting"` .	String
aws:ResourceTag/${TagKey}	The context key is formatted `"aws:ResourceTag/ tag-key":"tag-value "` where *tag-key* and *tag-value* are a tag key and value pair. Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.	String
aws:TagKeys	This context key is formatted `"aws:TagKeys":" tag-key"` where *tag-key* is a list of tag keys without values (for example, `["Dept","Cost-Center"]`). Checks the tag keys that are present in an AWS request.	String
ecs:ResourceTag/${TagKey}	The context key is formatted `"ecs:ResourceTag/ tag-key":"tag-value "` where *tag-key* and *tag-value* are a tag key and value pair.	String

Condition Key	Description	Evaluation Types
	Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.	
ecs:account-setting	he context key is formatted `"ecs:account-setting":"` *`account-setting`* `"` where *`account-setting`* is the name of the account setting.	String
ecs:auto-assign-public-ip	The context key is formatted `"ecs:auto-assign-public-ip":"` *`value`*`"` where *`value-`* is `"true"` or `"false"`.	String
ecs:capacity-provider	The context key is formatted `"ecs:capacity-provider":"` *`capacity-provider-arn`* `"` where *`capacity-provider-arn`* is the ARN for the capacity provider.	ARN, Null
ecs:cluster	The context key is formatted `"ecs:cluster":"` *`cluster-arn`* `"` where *`cluster-arn`* is the ARN for the Amazon ECS cluster.	ARN, Null
ecs:compute-compatability	The context key is formatted `"ecs:compute-compatability":"` *`compatability-type`* `"` where *`compatability-type`* is "FARGATE", "EC2" or "EXTERNAL".	String
ecs:container-instances	The context key is formatted `"ecs:container-instances":"` *`container-instance-arns`* `"` where *`container-instance-arns`* is one or more container instance ARNs.	ARN, Null
ecs:container-name	The context key is formatted `"ecs:container-name":"` *`container-name`* `"` where *`container-instance-`* is the name of an Amazon ECS container which is defined in the task definition.	String

Condition Key	Description	Evaluation Types
ecs:enable-execute-command	The context key is formatted `"ecs:enable-execute-command":"` *value*`"` where *value-* is `"true"` or `"false"`.	String
ecs:enable-service-connect	The context key is formatted `"ecs:enable-service-connect":"` *value*`"` where *value* is `"true"` or `"false"`.	String
ecs:enable-ebs-volumes	The context key is formatted `"ecs:enable-ebs-volumes":"` *value*`"` where *value* is `"true"` or `"false"`.	String
ecs:enable-managed-tags	The context key is formatted `"ecs:enable-managed-tags":"` *value*`"` where *value* is `"true"` or `"false"`.	String
ecs:enable-vpc-lattice	The context key is formatted `"ecs:vpc-lattice":"` *value*`"` where *value* is `"true"` or `"false"`.	String
ecs:fargate-ephemeral-storage-kms-key	The context key is formatted `"ecs:fargate-ephemeral-storage-kms-key":"` *key*`"` where *key* is the ID of the AWS KMS key.	String
ecs:namespace	The context key is formatted `"ecs:namespace":"` *namespace-arn* `"` where *namespace-arn* is the ARN for the AWS Cloud Map namespace.	ARN, Null
ecs:propagate-tags	The context key is formatted `"ecs:propagate-tags":"` *value*`"` where *value* is `"TASK_DEFINITION"` , `"SERVICE"` , or `"NONE"`.	String
ecs:service	The context key is formatted `"ecs:service":"` *service-arn* `"` where *service-arn* is the ARN for the Amazon ECS service.	ARN, Null

Condition Key	Description	Evaluation Types
ecs:task-definition	The context key is formatted `"ecs:task-definition":" task-definition-arn "` where *task-definition-arn* is the ARN for the Amazon ECS task definition.	ARN, Null
ecs:subnet	The context key is formatted `"ecs:subnet":" subnets"` where *subnets* are the subnets of a task or service that uses `awsvpc` mode.	String
ecs:task	The context key is formatted `"ecs:task":" task-arn"` where *task-arn* is the ARN for the Amazon ECS service.	ARN, Null
ecs:task-cpu	The context key is formatted `"ecs:task-cpu":"task-cpu"` where *task-cpu* is the task cpu, as an integer with 1024 = 1 vCPU.	Integer
ecs:task-memory	The context key is formatted `"ecs:task-memory":" task-memory "` where *task-memory* is the task memory in MiB.	Integer

To see a list of Amazon ECS condition keys, see Condition keys for Amazon Elastic Container Service in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon Elastic Container Service.

To view examples of Amazon ECS identity-based policies, see Identity-based policy examples for Amazon Elastic Container Service.

Access control lists (ACLs) in Amazon ECS

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Amazon ECS

> **Important**
>
> Amazon ECS supports attributes-based access control for all Amazon ECS resources. To determine whether you can use attributes to scope an action, use the Actions defined by Amazon ECS table in *Service Authorization Reference*. First verify that there is a resource in the **Resource** column. Then, use the **Condition keys** column to see the keys for the action/resource combination.

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging Amazon ECS resources, see Tagging Amazon ECS resources.

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Describing Amazon ECS services based on tags.

Using Temporary credentials with Amazon ECS

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Forward access sessions for Amazon ECS

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon ECS

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon ECS functionality. Edit service roles only when Amazon ECS provides guidance to do so.

Service-linked roles for Amazon ECS

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing Amazon ECS service-linked roles, see Using service-linked roles for Amazon ECS.

Identity-based policy examples for Amazon Elastic Container Service

By default, users and roles don't have permission to create or modify Amazon ECS resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon ECS, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon Elastic Container Service in the *Service Authorization Reference*.

Topics

- Amazon ECS task definition examples
- Run Amazon ECS Task Example
- Start Amazon ECS task example
- List and describe Amazon ECS task examples
- Create Amazon ECS service example
- Update Amazon ECS service example
- Describing Amazon ECS services based on tags
- Deny Amazon ECS Service Connect Namespace Override Example

Amazon ECS policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon ECS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and

functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Allow Amazon ECS users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
```

```
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS cluster examples

The following IAM policy allows permission to create and list clusters. The `CreateCluster` and `ListClusters` actions do not accept any resources, so the resource definition is set to `*` for all resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateCluster",
                "ecs:ListClusters"
            ],
            "Resource": ["*"]
        }
    ]
}
```

The following IAM policy allows permission to describe and delete a specific cluster. The `DescribeClusters` and `DeleteCluster` actions accept cluster ARNs as resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeClusters",
                "ecs:DeleteCluster"
            ],
            "Resource": ["arn:aws:ecs:us-east-1:<aws_account_id>:cluster/
<cluster_name>"]
```

```
        }
    ]
}
```

The following IAM policy can be attached to a user or group that would only allow that user or group to perform operations on a specific cluster.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:Describe*",
                "ecs:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "ecs:DeleteCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:ListContainerInstances",
                "ecs:RegisterContainerInstance",
                "ecs:SubmitContainerStateChange",
                "ecs:SubmitTaskStateChange"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
        },
        {
            "Action": [
                "ecs:DescribeContainerInstances",
                "ecs:DescribeTasks",
                "ecs:ListTasks",
                "ecs:UpdateContainerAgent",
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:RunTask"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
```

```
                "ArnEquals": {"ecs:cluster": "arn:aws:ecs:us-
east-1:<aws_account_id>:cluster/default"}
            }
        }
    ]
}
```

Amazon ECS container instance examples

Container instance registration is handled by the Amazon ECS agent, but there may be times where you want to allow a user to deregister an instance manually from a cluster. Perhaps the container instance was accidentally registered to the wrong cluster, or the instance was terminated with tasks still running on it.

The following IAM policy allows a user to list and deregister container instances in a specified cluster:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DeregisterContainerInstance",
                "ecs:ListContainerInstances"
            ],
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>"]
        }
    ]
}
```

The following IAM policy allows a user to describe a specified container instance in a specified cluster. To open this permission up to all container instances in a cluster, you can replace the container instance UUID with *.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": ["ecs:DescribeContainerInstances"],
            "Condition": {
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
            },
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:container-instance/
<cluster_name>/<container_instance_UUID>"]
        }
    ]
}
```

Amazon ECS task definition examples

Task definition IAM policies do not support resource-level permissions, but the following IAM policy allows a user to register, list, and describe task definitions:

If you use the console, you must add `CloudFormation: CreateStack` as an `Action`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RegisterTaskDefinition",
                "ecs:ListTaskDefinitions",
                "ecs:DescribeTaskDefinition"
            ],
            "Resource": ["*"]
        }
    ]
}
```

Run Amazon ECS Task Example

The resources for `RunTask` are task definitions. To limit which clusters a user can run task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to run any revision of a specific task definition on a specific cluster:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:RunTask"],
            "Condition": {
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
            },
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:task-definition/
<task_family>:*"]
        }
    ]
}
```

Start Amazon ECS task example

The resources for `StartTask` are task definitions. To limit which clusters and container instances a user can start task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to start any revision of a specific task definition on a specific cluster and specific container instance.

> **Note**
>
> For this example, when you call the `StartTask` API with the AWS CLI or another AWS SDK, you must specify the task definition revision so that the `Resource` mapping matches.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:StartTask"],
            "Condition": {
                "ArnEquals": {
```

```
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>",
                    "ecs:container-instances":
 ["arn:aws:ecs:<region>:<aws_account_id>:container-instance/<cluster_name>/
<container_instance_UUID>"]
                }
            },
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:task-definition/
<task_family>:*"]
        }
    ]
}
```

List and describe Amazon ECS task examples

The following IAM policy allows a user to list tasks for a specified cluster:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:ListTasks"],
            "Condition": {
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
            },
            "Resource": ["*"]
        }
    ]
}
```

The following IAM policy allows a user to describe a specified task in a specified cluster:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:DescribeTasks"],
            "Condition": {
                "ArnEquals": {"ecs:cluster":
 "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"}
```

```
            },
            "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:task/<cluster_name>/
<task_UUID>"]
        }
    ]
}
```

Create Amazon ECS service example

The following IAM policy allows a user to create Amazon ECS services in the AWS Management Console:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:Describe*",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "ecs:List*",
                "ecs:Describe*",
                "ecs:CreateService",
                "elasticloadbalancing:Describe*",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoles",
                "iam:ListGroups",
                "iam:ListUsers"
            ],
            "Resource": ["*"]
        }
    ]
}
```

Update Amazon ECS service example

The following IAM policy allows a user to update Amazon ECS services in the AWS Management Console:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:Describe*",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
                "ecs:List*",
                "ecs:Describe*",
                "ecs:UpdateService",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoles",
                "iam:ListGroups",
                "iam:ListUsers"
            ],
            "Resource": ["*"]
        }
    ]
}
```

Describing Amazon ECS services based on tags

You can use conditions in your identity-based policy to control access to Amazon ECS resources based on tags. This example shows how you might create a policy that allows describing your services. However, permission is granted only if the service tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeServices",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "*"
        },
        {
            "Sid": "ViewServiceIfOwner",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "arn:aws:ecs:*:*:service/*",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to describe an Amazon ECS service, the service must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

Deny Amazon ECS Service Connect Namespace Override Example

The following IAM policy denies a user from overriding the default Service Connect namespace in a service configuration. The default namespace is set in the cluster. However, you can override it in a service configuration. For consistency, consider setting all your new services to use the same namespace. Use the following context keys to require services to use a specific namespace. Replace the `<region>`, `<aws_account_id>`, `<cluster_name>` and `<namespace_id>` with your own in the following example.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateService",
                "ecs:UpdateService"
            ],
            "Condition": {
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/
<cluster_name>",
                    "ecs:namespace":
 "arn:aws:servicediscovery:<region>:<aws_account_id>:namespace/<namespace_id>"
                }
            },
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for Amazon Elastic Container Service

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies] that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies] in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

Amazon ECS and Amazon ECR provide several managed policies and trust relationships that you can attach to users, groups, roles, Amazon EC2 instances, and Amazon ECS tasks that allow differing levels of control over resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own policies. For more information about the Amazon ECR managed policies, see Amazon ECR managed policies.

AmazonECS_FullAccess

You can attach the `AmazonECS_FullAccess` policy to your IAM identities. This policy grants administrative access to Amazon ECS resources and grants an IAM identity (such as a user, group, or role) access to the AWS services that Amazon ECS is integrated with to use all of Amazon ECS features. Using this policy allows access to all of Amazon ECS features that are available in the AWS Management Console.

To view the permissions for this policy, see AmazonECS_FullAccess in the *AWS Managed Policy Reference*.

AmazonECSInfrastructureRolePolicyForVolumes

You can attach the `AmazonECSInfrastructureRolePolicyForVolumes` managed policy to your IAM entities.

The policy grants the permissions that are needed by Amazon ECS to make AWS API calls on your behalf. You can attach this policy to the IAM role that you provide with your volume configuration when you launch Amazon ECS tasks and services. The role allows Amazon ECS to manage volumes attached to your tasks. For more information, see Amazon ECS infrastructure IAM role.

To view the permissions for this policy, see AmazonECSInfrastructureRolePolicyForVolumes in the *AWS Managed Policy Reference*.

AmazonEC2ContainerServiceforEC2Role

You can attach the `AmazonEC2ContainerServiceforEC2Role` policy to your IAM identities. This policy grants administrative permissions that allow Amazon ECS container instances to make calls to AWS on your behalf. For more information, see Amazon ECS container instance IAM role.

Amazon ECS attaches this policy to a service role that allows Amazon ECS to perform actions on your behalf against Amazon EC2 instances or external instances.

To view the permissions for this policy, see AmazonEC2ContainerServiceforEC2Role in the *AWS Managed Policy Reference*.

Considerations

You should consider the following recommendations and considerations when using the `AmazonEC2ContainerServiceforEC2Role` managed IAM policy.

- Following the standard security advice of granting least privilege, you can modify the `AmazonEC2ContainerServiceforEC2Role` managed policy to fit your specific needs. If any of the permissions granted in the managed policy aren't needed for your use case, create a custom policy and add only the permissions that you require. For example, the `UpdateContainerInstancesState` permission is provided for Spot Instance draining. If that permission isn't needed for your use case, exclude it using a custom policy.
- Containers that are running on your container instances have access to all of the permissions that are supplied to the container instance role through instance metadata. We recommend that you limit the permissions in your container instance role to the minimal list of permissions that are provided in the managed `AmazonEC2ContainerServiceforEC2Role` policy. If the containers in your tasks need extra permissions that aren't listed, we recommend providing those tasks with their own IAM roles. For more information, see Amazon ECS task IAM role.

  You can prevent containers on the `docker0` bridge from accessing the permissions supplied to the container instance role. You can do this while still allowing the permissions that are provided by Amazon ECS task IAM role by running the following **iptables** command on your container instances. Containers can't query instance metadata with this rule in effect. This command assumes the default Docker bridge configuration and it doesn't work with containers that use the `host` network mode. For more information, see Network mode.

  ```
  sudo yum install -y iptables-services; sudo iptables --insert DOCKER USER 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
  ```

  You must save this **iptables** rule on your container instance for it to survive a reboot. For the Amazon ECS-optimized AMI, use the following command. For other operating systems, consult the documentation for that OS.

  - For the Amazon ECS-optimized Amazon Linux 2 AMI:

    ```
    sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
    ```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo service iptables save
```

AmazonEC2ContainerServiceEventsRole

You can attach the `AmazonEC2ContainerServiceEventsRole` policy to your IAM identities. This policy grants permissions that allow Amazon EventBridge (formerly CloudWatch Events) to run tasks on your behalf. This policy can be attached to the IAM role that's specified when you create scheduled tasks. For more information, see Amazon ECS EventBridge IAM Role.

To view the permissions for this policy, see AmazonEC2ContainerServiceEventsRole in the *AWS Managed Policy Reference*.

AmazonECSTaskExecutionRolePolicy

The `AmazonECSTaskExecutionRolePolicy` managed IAM policy grants the permissions that are needed by the Amazon ECS container agent and AWS Fargate container agents to make AWS API calls on your behalf. This policy can be added to your task execution IAM role. For more information, see Amazon ECS task execution IAM role.

To view the permissions for this policy, see AmazonECSTaskExecutionRolePolicy in the *AWS Managed Policy Reference*.

AmazonECSServiceRolePolicy

The `AmazonECSServiceRolePolicy` managed IAM policy enables Amazon Elastic Container Service to manage your cluster. This policy can be added to your AWSServiceRoleForECS service-linked role.

To view the permissions for this policy, see AmazonECSServiceRolePolicy in the *AWS Managed Policy Reference*.

`AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecur`

You can attach the `AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity` policy to your IAM entities. This policy grants administrative access to AWS Private Certificate Authority, Secrets Manager and other AWS Services required to manage Amazon ECS Service Connect TLS features on your behalf.

To view the permissions for this policy, see AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity in the *AWS Managed Policy Reference*.

AWSApplicationAutoscalingECSServicePolicy

You can't attach `AWSApplicationAutoscalingECSServicePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Application Auto Scaling to perform actions on your behalf. For more information, see Service-linked roles for Application Auto Scaling.

To view the permissions for this policy, see AWSApplicationAutoscalingECSServicePolicy in the *AWS Managed Policy Reference*.

AWSCodeDeployRoleForECS

You can't attach `AWSCodeDeployRoleForECS` to your IAM entities. This policy is attached to a service-linked role that allows CodeDeploy to perform actions on your behalf. For more information, see Create a service role for CodeDeploy in the *AWS CodeDeploy User Guide*.

To view the permissions for this policy, see AWSCodeDeployRoleForECS in the *AWS Managed Policy Reference*.

AWSCodeDeployRoleForECSLimited

You can't attach `AWSCodeDeployRoleForECSLimited` to your IAM entities. This policy is attached to a service-linked role that allows CodeDeploy to perform actions on your behalf. For more information, see Create a service role for CodeDeploy in the *AWS CodeDeploy User Guide*.

To view the permissions for this policy, see AWSCodeDeployRoleForECSLimited in the *AWS Managed Policy Reference*.

AmazonECSInfrastructureRolePolicyForLoadBalancers

You can attach the `AmazonECSInfrastructureRolePolicyForLoadBalancers` policy to your IAM entities. This policy grants permissions that allow Amazon ECS to manage Elastic Load Balancing resources on your behalf. The policy includes:

- Read-only permissions to describe listeners, rules, target groups, and target health
- Permissions to register and deregister targets with target groups
- Permissions to modify listeners for Application Load Balancers and Network Load Balancers
- Permissions to modify rules for Application Load Balancers

These permissions enable Amazon ECS to automatically manage load balancer configurations when services are created or updated, ensuring proper routing of traffic to your containers.

To view the permissions for this policy, see AmazonECSInfrastructureRolePolicyForLoadBalancers in the *AWS Managed Policy Reference*.

`AmazonECSInfrastructureRolePolicyForVpcLattice`

You can attach the `AmazonECSInfrastructureRolePolicyForVpcLattice` policy to your IAM entities. This policy Provides access to other AWS service resources required to manage VPC Lattice feature in Amazon ECS workloads on your behalf.

To view the permissions for this policy, see AmazonECSInfrastructureRolePolicyForVpcLattice in the *AWS Managed Policy Reference*.

Provides access to other AWS service resources required to manage VPC Lattice feature in Amazon ECS workloads on your behalf.

Amazon ECS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon ECS since this service started tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon ECS Document history page.

Change	Description	Date
Add permissions to the section called "AmazonEC2ContainerServiceforEC2Role"	The `AmazonEC2ContainerServiceforEC2Role` managed IAM policy was updated to include the `ecs:ListTagsForResource` permission. This permission allows the Amazon ECS agent to retrieve task and container instance tags through the task metadata endpoint (`${ECS_CONTAINER_ME`	August 4, 2025

Change	Description	Date
	TADATA_URI_V4}/tas kWithTags).	
Add permissions to AmazonECSInfrastructureRolePolicyForLoadBalancers.	The AmazonECSInfrastructureRolePolicyForLoadBalancers managed IAM policy was updated with new permissions for describing, deregistering, and registering target groups.	July 25, 2025
Add new the section called "AmazonECSInfrastructureRolePolicyForLoadBalancers" policy	Added new AmazonECSInfrastructureRolePolicyForLoadBalancers policy that provides access to other AWS service resources required to manage load balancers associated with Amazon ECS workloads.	July 15, 2025
Add permissions to AmazonECSServiceRolePolicy.	The AmazonECSServiceRolePolicy managed IAM policy was updated with new AWS Cloud Map permissions which Amazon ECS can update AWS Cloud Map service attributes for services that Amazon ECS manages.	July 15, 2025

Change	Description	Date
Add permissions to AmazonECSServiceRolePolicy	The `AmazonECSServiceRolePolicy` managed IAM policy was updated with new AWS Cloud Map permissions which Amazon ECS can update AWS Cloud Map service attributes for services that Amazon ECS manages.	June 24, 2025
Add permissions to AmazonECSInfrastructureRolePolicyForVolumes	The `AmazonECSInfrastructureRolePolicyForVolumes` policy has been updated to add the `ec2:DescribeInstances` permission. The permission helps prevent device name collision for Amazon EBS volumes that are attached to Amazon ECS tasks that run on the same container instance.	June 2, 2025
Add new AmazonECSInfrastructureRolePolicyForVpcLattice	Provides access to other AWS service resources required to manage VPC Lattice feature in Amazon ECS workloads on your behalf.	November 18, 2024
Add permissions to AmazonECSInfrastructureRolePolicyForVolumes	The `AmazonECSInfrastructureRolePolicyForVolumes` policy has been updated to allow customers to create an Amazon EBS volume from a snapshot.	October 10, 2024

Change	Description	Date
Added permissions to the section called "AmazonECS_FullAccess"	The `AmazonECS_FullAccess` policy was updated to add `iam:PassRole` permissions for IAM roles for a role named `ecsInfrastructureRole` . This is the default IAM role created by the AWS Management Console that is intended to be used as an ECS infrastructure role that allows Amazon ECS to manage Amazon EBS volumes attached to ECS tasks.	August 13, 2024
Add new AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity policy	Added new AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity policy that provides administrative access to AWS KMS, AWS Private Certificate Authority, Secrets Manager and enables Amazon ECS Service Connect TLS features to work properly.	January 22, 2024
Add new policy AmazonECSInfrastructureRolePolicyForVolumes	The `AmazonECSInfrastructureRolePolicyForVolumes` policy was added. The policy grants the permissions that are needed by Amazon ECS to make AWS API calls to manage Amazon EBS volumes associated with Amazon ECS workloads.	January 11, 2024

Change	Description	Date
Add permissions to [AmazonECSServiceRolePolicy](#)	The `AmazonECSServiceRolePolicy` managed IAM policy was updated with new `events` permissions and additional `autoscaling` and `autoscaling-plans` permissions.	December 4, 2023
Add permissions to [AmazonEC2ContainerServiceEventsRole](#)	The `AmazonECSServiceRolePolicy` managed IAM policy was updated to allow access to the AWS Cloud Map `DiscoverInstancesRevision` API operation.	October 4, 2023
Add permissions to [AmazonEC2ContainerServiceforEC2Role](#)	The `AmazonEC2ContainerServiceforEC2Role` policy was modified to add the `ecs:TagResource` permission, which includes a condition that limits the permission only to newly created clusters and registered container instances.	March 6, 2023

Change	Description	Date
Add permissions to the section called “AmazonECS_FullAccess”	The AmazonECS_FullAccess policy was modified to add the elasticloadbalancing:AddTags permission, which includes a condition that limits the permission only to newly created load balancers, target groups, rules, and listeners created. This permission doesn't allow tags to be added to any already created Elastic Load Balancing resources.	January 4, 2023
Amazon ECS started tracking changes	Amazon ECS started tracking changes for its AWS managed policies.	June 8, 2021

Phased out AWS managed IAM policies for Amazon Elastic Container Service

The following AWS managed IAM policies are phased out. These policies are now replaced by the updated policies. We recommend that you update your users or roles to use the updated policies.

AmazonEC2ContainerServiceFullAccess

Important

The `AmazonEC2ContainerServiceFullAccess` managed IAM policy was phased out as of January 29, 2021, in response to a security finding with the `iam:passRole` permission. This permission grants access to all resources including credentials to roles in the account. Now that the policy is phased out, you can't attach the policy to any new users or roles. Any users or roles that already have the policy attached can continue using it. However, we recommend that you update your users or roles to use the

AmazonECS_FullAccess managed policy instead. For more information, see Migrating to the AmazonECS_FullAccess managed policy.

AmazonEC2ContainerServiceRole

> **Important**
>
> The AmazonEC2ContainerServiceRole managed IAM policy is phased out. It's now replaced by the Amazon ECS service-linked role. For more information, see Using service-linked roles for Amazon ECS.

AmazonEC2ContainerServiceAutoscaleRole

> **Important**
>
> The AmazonEC2ContainerServiceAutoscaleRole managed IAM policy is phased out. It's now replaced by the Application Auto Scaling service-linked role for Amazon ECS. For more information, see Service-linked roles for Application Auto Scaling in the *Application Auto Scaling User Guide*.

Migrating to the AmazonECS_FullAccess managed policy

The AmazonEC2ContainerServiceFullAccess managed IAM policy was phased out on January 29, 2021, in response to a security finding with the iam:passRole permission. This permission grants access to all resources including credentials to roles in the account. Now that the policy is phased out, you can't attach the policy to any new groups, users, or roles. Any groups, users, or roles that already have the policy attached can continue using it. However, we recommend that you update your groups, users, or roles to use the AmazonECS_FullAccess managed policy instead.

The permissions that are granted by the AmazonECS_FullAccess policy include the complete list of permissions that are necessary to use ECS as an administrator. If you currently use permissions that are granted by the AmazonEC2ContainerServiceFullAccess policy that aren't in the AmazonECS_FullAccess policy, you can add them to an inline policy statement. For more information, see AWS managed policies for Amazon Elastic Container Service.

Use the following steps to determine if you have any groups, users, or roles that are currently using the `AmazonEC2ContainerServiceFullAccess` managed IAM policy. Then, update them to detach the earlier policy and attach the `AmazonECS_FullAccess` policy.

To update a group, user, or role to use the AmazonECS_FullAccess policy (AWS Management Console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies** and search for and select the `AmazonEC2ContainerServiceFullAccess` policy.
3. Choose the **Policy usage** tab that displays any IAM role that's currently using this policy.
4. For each IAM role that's currently using the `AmazonEC2ContainerServiceFullAccess` policy, select the role and use the following steps to detach the phased out policy and attach the `AmazonECS_FullAccess` policy.

   a. On the **Permissions** tab, choose the **X** next to the **AmazonEC2ContainerServiceFullAccess** policy.

   b. Choose **Add permissions**.

   c. Choose **Attach existing policies directly**, search for and select the **AmazonECS_FullAccess** policy, and then choose **Next: Review**.

   d. Review the changes and then choose **Add permissions**.

   e. Repeat these steps for each group, user, or role that's using the `AmazonEC2ContainerServiceFullAccess` policy.

To update a group, user, or role to use the `AmazonECS_FullAccess` policy (AWS CLI)

1. Use the generate-service-last-accessed-details command to generate a report that includes details about when the phased out policy was last used.

```
aws iam generate-service-last-accessed-details \
     --arn arn:aws:iam::aws:policy/AmazonEC2ContainerServiceFullAccess
```

   Example output:

```
{
    "JobId": "32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE"
```

```
}
```

2. Use the job ID from the previous output with the get-service-last-accessed-details command to retrieve the last accessed report of the service. This report displays the Amazon Resource Name (ARN) of the IAM entities that last used the phased out policy.

```
aws iam get-service-last-accessed-details \
      --job-id 32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE
```

3. Use one of the following commands to detach the `AmazonEC2ContainerServiceFullAccess` policy from a group, user, or role.

   - detach-group-policy
   - detach-role-policy
   - detach-user-policy

4. Use one of the following commands to attach the `AmazonECS_FullAccess` policy to a group, user, or role.

   - attach-group-policy
   - attach-role-policy
   - attach-user-policy

Using service-linked roles for Amazon ECS

Amazon Elastic Container Service uses AWS Identity and Access Management (IAM)service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon ECS. The service-linked role is predefined by Amazon ECS and includes all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon ECS easier because you don't have to manually add the necessary permissions. Amazon ECS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon ECS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon ECS

Amazon ECS uses the service-linked role named **AWSServiceRoleForECS**.

The AWSServiceRoleForECS service-linked role trusts the following services to assume the role:

- `ecs.amazonaws.com`

The role permissions policy named AmazonECSServiceRolePolicy allows Amazon ECS to complete the following actions on the specified resources:

- Action: When using the `awsvpc` network mode for your Amazon ECS tasks, Amazon ECS manages the lifecycle of the elastic network interfaces associated with the task. This also includes tags that Amazon ECS adds to your elastic network interfaces.
- Action: When using a load balancer with your Amazon ECS service, Amazon ECS manages the registration and deregistration of resources with the load balancer.
- Action: When using Amazon ECS service discovery, Amazon ECS manages the required Route 53 and AWS Cloud Map resources for service discovery to work.
- Action: When using Amazon ECS service auto scaling, Amazon ECS manages the required Auto Scaling resources.
- Action: Amazon ECS creates and manages CloudWatch alarms and log streams that assist in the monitoring of your Amazon ECS resources.
- Action: When using Amazon ECS Exec, Amazon ECS manages the permissions needed to start Amazon ECS Exec sessions to your tasks.
- Action: When using Amazon ECS Service Connect, Amazon ECS manages the required AWS Cloud Map resources to use the feature.
- Action: When using Amazon ECS capacity providers, Amazon ECS manages the permissions required to modify the Auto Scaling group and its Amazon EC2 instances.
- Action: Amazon ECS can update AWS Cloud Map service attributes for services that Amazon ECS manages.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon ECS

In most cases you don't need to manually create a service-linked role. When you create a cluster or create or update a service in the AWS Management Console, the AWS CLI, or the AWS API, Amazon ECS creates the service-linked role for you. If you do not see the **AWSServiceRoleForECS** role after you create a cluster, perform the following to fix the issue:

- Verify and configure the permissions to allow Amazon ECS to create, edit, or delete a service-linked role on your behalf. For more information, see Service-linked role permissions in the *IAM User Guide*.
- Retry the cluster creation operation, or manually create the service-linked role.

  You can use the IAM console to create the **AWSServiceRoleForECS** service-linked role. In the AWS CLI or the AWS API, create a service-linked role with the `ecs.amazonaws.com` service name. For more information, see Creating a service-linked role in the *IAM User Guide*.

> **Important**
>
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a cluster or create or update a service, Amazon ECS creates the service-linked role for you again.

If you delete this service-linked role, you can use the same IAM process to create the role again.

Editing a service-linked role for Amazon ECS

Amazon ECS doesn't allow you to edit the AWSServiceRoleForECS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Update a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon ECS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
>
> If the Amazon ECS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To check whether the service-linked role has an active session

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles** and choose the AWSServiceRoleForECS name (not the check box).
3. On the **Summary** page, choose **Access Advisor** and review recent activity for the service-linked role.

   > **Note**
   >
   > If you are unsure whether Amazon ECS is using the AWSServiceRoleForECS role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

To remove Amazon ECS resources used by the AWSServiceRoleForECS service-linked role

You must delete all Amazon ECS clusters in all AWS Regions before you can delete the AWSServiceRoleForECS role.

1. Scale all Amazon ECS services down to a desired count of 0 in all regions, and then delete the services. For more information, see Updating an Amazon ECS service and Deleting an Amazon ECS service using the console.
2. Force deregister all container instances from all clusters in all regions. For more information, see Deregistering an Amazon ECS container instance.
3. Delete all Amazon ECS clusters in all regions. For more information, see Deleting an Amazon ECS cluster.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForECS service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported regions for Amazon ECS service-linked roles

Amazon ECS supports using service-linked roles in all of the regions where the service is available. For more information, see AWS regions and endpoints.

IAM roles for Amazon ECS

An IAM role is an IAM identity that you can create in your account that has specific permissions. In Amazon ECS, you can create roles to grant permissions to Amazon ECS resource such as containers or services.

The roles Amazon ECS requires depend on the task definition launch type and the features that you use. Use the following table to determine which IAM roles you need for Amazon ECS.

Role	Definition	When required	More information
Task execution role	This role allows Amazon ECS to use other AWS services on your behalf.	Your task is hosted on AWS Fargate or on external instances and: • pulls a container image from an Amazon ECR private repository. • pulls a container image from an Amazon ECR private repositor y in a different account from the account that runs the task.	Amazon ECS task execution IAM role

Role	Definition	When required	More information
		• sends container logs to CloudWatch Logs using the `awslogs` log driver. Your task is hosted on either AWS Fargate or Amazon EC2 instances and: • uses private registry authentication. • uses Runtime Monitoring. • the task definition references sensitive data using Secrets Manager secrets or AWS Systems Manager Parameter Store parameters.	
Task role	This role allows your application code (on the container) to use other AWS services.	Your application accesses other AWS services, such as Amazon S3.	Amazon ECS task IAM role
Container instance role	This role allows your EC2 instances or external instances to register with the cluster.	Your task is hosted on Amazon EC2 instances or an external instance.	Amazon ECS container instance IAM role

Role	Definition	When required	More information
Amazon ECS Anywhere role	This role allows your external instances to access AWS APIs.	Your task is hosted on external instances.	Amazon ECS Anywhere IAM role
Amazon ECS infrastructure for load balancers role	This role allows Amazon ECS to manage load balancer resources in your clusters on your behalf for blue/green deployments.	You want to use Amazon ECS blue/green deployments.	Amazon ECS infrastructure IAM role for load balancers
Amazon ECS CodeDeploy role	This role allows CodeDeploy to make updates to your services.	You use the CodeDeploy blue/green deployment type to deploy services.	Amazon ECS CodeDeploy IAM Role
Amazon ECS EventBridge role	This role allows EventBridge to make updates to your services.	You use the EventBridge rules and targets to schedule your tasks.	Amazon ECS EventBridge IAM Role

Role	Definition	When required	More information
Amazon ECS infrastru cture role	This role allows Amazon ECS to manage infrastru cture resources in your clusters.	• You want to attach Amazon EBS volumes to your Fargate or EC2 launch type Amazon ECS tasks. The infrastru cture role allows Amazon ECS to manage Amazon EBS volumes for your tasks. • You want to use Transport Layer Security (TLS) to encrypt traffic between your Amazon ECS Service Connect services. • You want to create VPC Lattice target groups.	Amazon ECS infrastru cture IAM role

Best practices for IAM roles in Amazon ECS

The roles Amazon ECS requires depend on the task definition launch type and the features that you use. We recommend that you create separate roles in the table instead of sharing roles.

Role	Definition	When required	More information
Task execution role	This role allows Amazon ECS to use	Your task is hosted on AWS Fargate or	Amazon ECS task execution IAM role

Role	Definition	When required	More information
	other AWS services on your behalf.	on external instances and: • pulls a container image from an Amazon ECR private repository. • pulls a container image from an Amazon ECR private repository in a different account from the account that runs the task. • sends container logs to CloudWatch Logs using the `awslogs` log driver. Your task is hosted on either AWS Fargate or Amazon EC2 instances and: • uses private registry authentication. • uses Runtime Monitoring. • the task definition references sensitive data using Secrets	

Role	Definition	When required	More information
		Manager secrets or AWS Systems Manager Parameter Store parameters.	
Task role	This role allows your application code (on the container) to use other AWS services.	Your application accesses other AWS services, such as Amazon S3.	Amazon ECS task IAM role
Container instance role	This role allows your EC2 instances or external instances to register with the cluster.	Your task is hosted on Amazon EC2 instances or an external instance.	Amazon ECS container instance IAM role
Amazon ECS Anywhere role	This role allows your external instances to access AWS APIs.	Your task is hosted on external instances.	Amazon ECS Anywhere IAM role
Amazon ECS CodeDeploy role	This role allows CodeDeploy to make updates to your services.	You use the CodeDeploy blue/green deploymen t type to deploy services.	Amazon ECS CodeDeploy IAM Role
Amazon ECS EventBridge role	This role allows EventBridge to make updates to your services.	You use the EventBrid ge rules and targets to schedule your tasks.	Amazon ECS EventBridge IAM Role

Role	Definition	When required	More information
Amazon ECS infrastru cture role	This role allows Amazon ECS to manage infrastru cture resources in your clusters.	• You want to attach Amazon EBS volumes to your Fargate or EC2 launch type Amazon ECS tasks. The infrastru cture role allows Amazon ECS to manage Amazon EBS volumes for your tasks. • You want to use Transport Layer Security (TLS) to encrypt traffic between your Amazon ECS Service Connect services. • You want to create VPC Lattice target groups.	[Amazon ECS infrastru cture IAM role]()

Task role

We recommend that you assign a task role. Its role can be distinguished from the role of the Amazon EC2 instance that it's running on. Assigning each task a role aligns with the principle of least privileged access and allows for greater granular control over actions and resources.

When you add a task role to a task definition, the Amazon ECS container agent automatically creates a token with a unique credential ID (for example, `12345678-90ab-cdef-1234-567890abcdef`) for the task. This token and the role credentials are then added to the agent's internal cache. The agent populates the environment variable

`AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` in the container with the URI of the credential ID (for example, `/v2/credentials/12345678-90ab-cdef-1234-567890abcdef`).

You can manually retrieve the temporary role credentials from inside a container by appending the environment variable to the IP address of the Amazon ECS container agent and running the `curl` command on the resulting string.

```
curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

The expected output is as follows:

```
{
 "RoleArn": "arn:aws:iam::123456789012:role/SSMTaskRole-SSMFargateTaskIAMRole-
DASWWSF2WGD6",
 "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
 "Token": "IQoJb3JpZ2luX2VjEEM/Example==",
 "Expiration": "2021-01-16T00:51:53Z"
}
```

Newer versions of the AWS SDKs automatically fetch these credentials from the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable when making AWS API calls. For information about how to renew credentials, see [Renewing AWS credentials] on rePost.

The output includes an access key-pair consisting of a secret access key ID and a secret key which your application uses to access AWS resources. It also includes a token that AWS uses to verify that the credentials are valid. By default, credentials assigned to tasks using task roles are valid for six hours. After that, they are automatically rotated by the Amazon ECS container agent.

Task execution role

The task execution role is used to grant the Amazon ECS container agent permission to call specific AWS API actions on your behalf. For example, when you use AWS Fargate, Fargate needs an IAM role that allows it to pull images from Amazon ECR and write logs to CloudWatch Logs. An IAM role is also required when a task references a secret that's stored in AWS Secrets Manager, such as an image pull secret.

> **Note**
>
> If you're pulling images as an authenticated user, you're less likely to be impacted by the changes that occurred to Docker Hub's pull rate limits. For more information see, Private registry authentication for container instances.
>
> By using Amazon ECR and Amazon ECR Public, you can avoid the limits imposed by Docker. If you pull images from Amazon ECR, this also helps shorten network pull times and reduces data transfer changes when traffic leaves your VPC.

> **Important**
>
> When you use Fargate, you must authenticate to a private image registry using `repositoryCredentials`. It's not possible to set the Amazon ECS container agent environment variables `ECS_ENGINE_AUTH_TYPE` or `ECS_ENGINE_AUTH_DATA` or modify the `ecs.config` file for tasks hosted on Fargate. For more information, see Private registry authentication for tasks.

Container instance role

The `AmazonEC2ContainerServiceforEC2Role` managed IAM policy includes the following permissions. Following the standard security advice of granting least privilege, the `AmazonEC2ContainerServiceforEC2Role` managed policy can be used as a guide. If you don't need any of the permissions that are granted in the managed policy for your use case, create a custom policy and add only the permissions that you need.

- `ec2:DescribeTags` – (Optional) Allows a principal to describe the tags that are associated with an Amazon EC2 instance. This permission is used by the Amazon ECS container agent to support resource tag propagation. For more information, see How resources are tagged.
- `ecs:CreateCluster` – (Optional) Allows a principal to create an Amazon ECS cluster. This permission is used by the Amazon ECS container agent to create a `default` cluster, if one doesn't already exist.
- `ecs:DeregisterContainerInstance` – (Optional) Allows a principal to deregister an Amazon ECS container instance from a cluster. The Amazon ECS container agent doesn't call this API operation, but this permission remains to help ensure backwards compatibility.

- `ecs:DiscoverPollEndpoint` – (Required) This action returns endpoints that the Amazon ECS container agent uses to poll for updates.
- `ecs:Poll` – (Required) Allows the Amazon ECS container agent to communicate with the Amazon ECS control plane to report task state changes.
- `ecs:RegisterContainerInstance` – (Required) Allows a principal to register a container instance with a cluster. This permission is used by the Amazon ECS container agent to register the Amazon EC2 instance with a cluster and to support resource tag propagation.
- `ecs:StartTelemetrySession` – (Optional) Allows the Amazon ECS container agent to communicate with the Amazon ECS control plane to report health information and metrics for each container and task.

  Although this permission is not required, we recommend that you add it to allowthe container instance metrics to start scale actions and also receive reports related to health check commands.
- `ecs:TagResource` – (Optional) Allows the Amazon ECS container agent to tag cluster on creation and to tag container instances when they are registered to a cluster.
- `ecs:UpdateContainerInstancesState` – Allows a principal to modify the status of an Amazon ECS container instance. This permission is used by the Amazon ECS container agent for Spot Instance draining.
- `ecs:Submit*` – (Required) This includes the `SubmitAttachmentStateChanges`, `SubmitContainerStateChange`, and `SubmitTaskStateChange` API actions. They're used by the Amazon ECS container agent to report state changes for each resource to the Amazon ECS control plane. The `SubmitContainerStateChange` permission is no longer used by the Amazon ECS container agent but remains to help ensure backwards compatibility.
- `ecr:GetAuthorizationToken` – (Optional) Allows a principal to retrieve an authorization token. The authorization token represents your IAM authentication credentials and can be used to access any Amazon ECR registry that the IAM principal has access to. The authorization token received is valid for 12 hours.
- `ecr:BatchCheckLayerAvailability` – (Optional) When a container image is pushed to an Amazon ECR private repository, each image layer is checked to verify if it's already pushed. If it is, then the image layer is skipped.
- `ecr:GetDownloadUrlForLayer` – (Optional) When a container image is pulled from an Amazon ECR private repository, this API is called once for each image layer that's not already cached.

- `ecr:BatchGetImage` – (Optional) When a container image is pulled from an Amazon ECR private repository, this API is called once to retrieve the image manifest.
- `logs:CreateLogStream` – (Optional) Allows a principal to create a CloudWatch Logs log stream for a specified log group.
- `logs:PutLogEvents` – (Optional) Allows a principal to upload a batch of log events to a specified log stream.

Service-linked roles

You can use the service-linked role for Amazon ECS to grant the Amazon ECS service permission to call other service APIs on your behalf. Amazon ECS needs the permissions to create and delete network interfaces, register, and de-register targets with a target group. It also needs the necessary permissions to create and delete scaling policies. These permissions are granted through the service-linked role. This role is created on your behalf the first time that you use the service.

> **Note**
>
> If you inadvertently delete the service-linked role, you can recreate it. For instructions, see Create the service-linked role.

Roles recommendations

We recommend that you do the following when setting up your task IAM roles and policies.

Block access to Amazon EC2 metadata

When you run your tasks on Amazon EC2 instances, we strongly recommend that you block access to Amazon EC2 metadata to prevent your containers from inheriting the role assigned to those instances. If your applications have to call an AWS API action, use IAM roles for tasks instead.

To prevent tasks running in **bridge** mode from accessing Amazon EC2 metadata, run the following command or update the instance's user data. For more instruction on updating the user data of an instance, see this AWS Support Article. For more information about the task definition bridge mode, see task definition network mode.

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 1 --in-interface
 docker+ --destination 169.254.169.254/32 --jump DROP
```

For this change to persist after a reboot, run the following command that's specific for your Amazon Machine Image (AMI):

- Amazon Linux 2

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now
 iptables
```

- Amazon Linux

```
sudo service iptables save
```

For tasks that use `awsvpc` network mode, set the environment variable `ECS_AWSVPC_BLOCK_IMDS` to `true` in the `/etc/ecs/ecs.config` file.

You should set the `ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST` variable to `false` in the `ecs-agent config` file to prevent the containers that are running within the `host` network from accessing the Amazon EC2 metadata.

Use the `awsvpc` network mode

Use the network `awsvpc` network mode to restrict the flow of traffic between different tasks or between your tasks and other services that run within your Amazon VPC. This adds an additional layer of security. The `awsvpc` network mode provides task-level network isolation for tasks that run on Amazon EC2. It is the default mode on AWS Fargate. it's the only network mode that you can use to assign a security group to tasks.

Use last accessed information to refine roles

We recommend that you remove any actions that were never used or haven't been used for some time. This prevents unwanted access from happening. To do this, review last accessed information provided by IAM, and then remove actions that were never used or haven't been used recently. You can do this by following the following steps.

Run the following command to generate a report showing the last access information for the referenced policy:

```
aws iam generate-service-last-accessed-details --arn arn:aws:iam::123456789012:policy/
ExamplePolicy1
```

use the `JobId` that was in the output to run the following command. After you do this, you can view the results of the report.

```
aws iam get-service-last-accessed-details --job-id 98a765b4-3cde-2101-2345-example678f9
```

For more information, see Refine permissions in AWS using last accessed information.

Monitor AWS CloudTrail for suspicious activity

You can monitor AWS CloudTrail for any suspicious activity. Most AWS API calls are logged to AWS CloudTrail as events. They are analyzed by AWS CloudTrail Insights, and you're alerted of any suspicious behavior that's associated with `write` API calls. This might include a spike in call volume. These alerts include such information as the time the unusual activity occurred and the top identity ARN that contributed to the APIs.

You can identify actions that are performed by tasks with an IAM role in AWS CloudTrail by looking at the event's `userIdentity` property. In the following example, the `arn` includes of the name of the assumed role, `s3-write-go-bucket-role`, followed by the name of the task, `7e9894e088ad416eb5cab92afExample`.

```
"userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROA36C6WWEJ2YEXAMPLE:7e9894e088ad416eb5cab92afExample",
    "arn": "arn:aws:sts::123456789012:assumed-role/s3-write-go-bucket-
role/7e9894e088ad416eb5cab92afExample",
    ...
}
```

> **Note**
>
> When tasks that assume a role are run on Amazon EC2 container instances, a request is logged by Amazon ECS container agent to the audit log of the agent that's located at an address in the `/var/log/ecs/audit.log.YYYY-MM-DD-HH` format. For more information, see Task IAM Roles Log and Logging Insights Events for Trails.

Amazon ECS task execution IAM role

The task execution role grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your behalf. The task execution IAM role is required depending on the requirements of your task. You can have multiple task execution roles for different purposes and services associated with your account.

> **Note**
>
> These permissions are made available to the agent running on your instance by Amazon ECS periodically sending it the role's temporary credentials, but they aren't directly accessible by the containers in the task. For the IAM permissions that your application code inside the container needs to run, see Amazon ECS task IAM role.

The following are common use cases for a task execution IAM role:

- Your task is hosted on AWS Fargate or on an external instance and:
  - pulls a container image from an Amazon ECR private repository.
  - pulls a container image from an Amazon ECR private repository in a different account from the account that runs the task.
  - sends container logs to CloudWatch Logs using the `awslogs` log driver. For more information, see Send Amazon ECS logs to CloudWatch .
- Your tasks are hosted on either AWS Fargate or Amazon EC2 instances and:
  - uses private registry authentication. For more information, see Private registry authentication permissions.
  - uses Runtime Monitoring.
  - the task definition references sensitive data using Secrets Manager secrets or AWS Systems Manager Parameter Store parameters. For more information, see Secrets Manager or Systems Manager permissions.

> **Note**
>
> The task execution role is supported by Amazon ECS container agent version 1.16.0 and later.

Amazon ECS provides the managed policy named `AmazonECSTaskExecutionRolePolicy` which contains the permissions the common use cases described above require. For more information, see AmazonECSTaskExecutionRolePolicy in the *AWS Managed Policy Reference Guide*. It might be necessary to add inline policies to your task execution role for special use cases

The Amazon ECS console creates a task execution role. You can manually attach the managed IAM policy for tasks to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. You can use IAM console search to search for `ecsTaskExecutionRole` and see if your account already has the task execution role. For more information, see IAM console search in the *IAM user guide*.

If you pull images as an authenticated user, you're less likely to be impacted by the changes that occurred to Docker Hub usage and limits. For more information see, Private registry authentication for container instances.

By using Amazon ECR and Amazon ECR Public, you can avoid the limits imposed by Docker. If you pull images from Amazon ECR, this also helps shorten network pull times and reduces data transfer changes when traffic leaves your VPC.

When you use Fargate, you must authenticate to a private image registry using `repositoryCredentials`. It's not possible to set the Amazon ECS container agent environment variables `ECS_ENGINE_AUTH_TYPE` or `ECS_ENGINE_AUTH_DATA` or modify the `ecs.config` file for tasks hosted on Fargate. For more information, see Private registry authentication for tasks.

Creating the task execution role

If your account doesn't already have a task execution role, use the following steps to create the role.

AWS Management Console

To create the service role for Elastic Container Service (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **Elastic Container Service**, and then choose the **Elastic Container Service Task** use case.
5. Choose **Next**.

6. In the **Add permissions** section, search for **AmazonECSTaskExecutionRolePolicy**, then select the policy.
7. Choose **Next**.
8. For **Role name**, enter **ecsTaskExecutionRole**.
9. Review the role, and then choose **Create role**.

AWS CLI

Replace all *user input* with your own information.

1. Create a file named `ecs-tasks-trust-policy.json` that contains the trust policy to use for the IAM role. The file should contain the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Create an IAM role named `ecsTaskExecutionRole` using the trust policy created in the previous step.

```
aws iam create-role \
      --role-name ecsTaskExecutionRole \
      --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. Attach the AWS managed `AmazonECSTaskExecutionRolePolicy` policy to the `ecsTaskExecutionRole` role.

```
aws iam attach-role-policy \
      --role-name ecsTaskExecutionRole \
```

```
    --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy
```

After you create the role, add additional permissions to the role for the following features.

Feature	Additional permissions
Pull container images from private registries outside of AWS (such as Docker Hub, Quay.io, or your own private registry) using Secrets Manager credentials	Private registry authentication permissions
Pass sensitive data with Systems Manager or Secrets Manager	Secrets Manager or Systems Manager permissions
Have Fargate tasks pull Amazon ECR images over interface endpoints	Fargate tasks pulling Amazon ECR images over interface endpoints permissions
Host configuration files in an Amazon S3 bucket	Amazon S3 file storage permissions
Configure Container Insights to view Amazon ECS lifecycle events	Permissions required for enabling Amazon ECS lifecycle events inContainer Insights
View Amazon ECS lifecycle events in Container Insights	Permissions required to view Amazon ECS lifecycle events in Container Insights

Private registry authentication permissions

Private registry authentication allows your Amazon ECS tasks to pull container images from private registries outside of AWS (such as Docker Hub, Quay.io, or your own private registry) that require authentication credentials. This feature uses Secrets Manager to securely store your registry credentials, which are then referenced in your task definition using the `repositoryCredentials` parameter.

For more information about configuring private registry authentication, see Using non-AWS container images in Amazon ECS.

To provide access to the secrets that contain your private registry credentials, add the following permissions as an inline policy to the task execution role. For more information, see Adding and Removing IAM Policies.

- `secretsmanager:GetSecretValue`—Required to retrieve the private registry credentials from Secrets Manager.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The Amazon Resource Name (ARN) for your custom key must be added as a resource.

The following is an example inline policy that adds the permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:secret_name",
                "arn:aws:kms:<region>:<aws_account_id>:key/key_id"
            ]
        }
    ]
}
```

Secrets Manager or Systems Manager permissions

The permission to allow the container agent to pull the necessary AWS Systems Manager or Secrets Manager resources. For more information, see Pass sensitive data to an Amazon ECS container.

Using Secrets Manager

To provide access to the Secrets Manager secrets that you create, manually add the following permission to the task execution role. For information about how to manage permissions, see Adding and Removing IAM identity permissions in the *IAM User Guide*.

- `secretsmanager:GetSecretValue`– Required if you are referencing a Secrets Manager secret. Adds the permission to retrieve the secret from Secrets Manager.

The following example policy adds the required permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
      ]
    }
  ]
}
```

Using Systems Manager

> **Important**
>
> For tasks that use the EC2 launch type, you must use the ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see Amazon ECS container agent configuration.

To provide access to the Systems Manager Parameter Store parameters that you create, manually add the following permissions as a policy to the task execution role. For information about how to manage permissions, see Adding and Removing IAM identity permissions in the *IAM User Guide*.

- `ssm:GetParameters` — Required if you are referencing a Systems Manager Parameter Store parameter in a task definition. Adds the permission to retrieve Systems Manager parameters.
- `secretsmanager:GetSecretValue` — Required if you are referencing a Secrets Manager secret either directly or if your Systems Manager Parameter Store parameter is referencing a Secrets Manager secret in a task definition. Adds the permission to retrieve the secret from Secrets Manager.

- `kms:Decrypt` — Required only if your secret uses a customer managed key and not the default key. The ARN for your custom key should be added as a resource. Adds the permission to decrypt the customer managed key .

The following example policy adds the required permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameters",
        "secretsmanager:GetSecretValue",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:ssm:region:aws_account_id:parameter/parameter_name",
        "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name",
        "arn:aws:kms:region:aws_account_id:key/key_id"
      ]
    }
  ]
}
```

Fargate tasks pulling Amazon ECR images over interface endpoints permissions

When launching tasks that use the Fargate launch type that pull images from Amazon ECR when Amazon ECR is configured to use an interface VPC endpoint, you can restrict the tasks access to a specific VPC or VPC endpoint. Do this by creating a task execution role for the tasks to use that use IAM condition keys.

Use the following IAM global condition keys to restrict access to a specific VPC or VPC endpoint. For more information, see [AWS Global Condition Context Keys](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_condition-keys.html).

- `aws:SourceVpc`—Restricts access to a specific VPC. You can restrict the VPC to the VPC that hosts the task and endpoint.
- `aws:SourceVpce`—Restricts access to a specific VPC endpoint.

The following task execution role policy provides an example for adding condition keys:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "ecr:GetAuthorizationToken",
                    "logs:CreateLogStream",
                    "logs:PutLogEvents"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "ecr:BatchCheckLayerAvailability",
                    "ecr:GetDownloadUrlForLayer",
                    "ecr:BatchGetImage"
                ],
                "Resource": "arn:aws:ecr:*:*:repository/*",
                "Condition": {
                    "StringEquals": {
                            "aws:sourceVpce": "vpce-0123456789abcdef0"
                    }
                }
            }
    ]
}
```

Amazon ECR permissions

The following permissions are required when you need to pull container images from Amazon ECR private repositories. The task execution role should have these permissions to allow the Amazon ECS container and Fargate agents to pull container images on your behalf. For basic ECS implementations, these permissions should be added to the task execution role rather than the task IAM role.

The Amazon ECS task execution role managed policy (`AmazonECSTaskExecutionRolePolicy`) includes the necessary permissions for pulling images from Amazon ECR. If you're using the managed policy, you don't need to add these permissions separately.

If you're creating a custom policy, include the following permissions to allow pulling images from Amazon ECR:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

Note that these permissions are different from the permissions that might be required in the task IAM role if your application code needs to interact with Amazon ECR APIs directly. For information about task IAM role permissions for Amazon ECR, see Amazon ECR permissions.

Amazon S3 file storage permissions

When you specify a configuration file that's hosted in Amazon S3, the task execution role must include the `s3:GetObject` permission for the configuration file and the `s3:GetBucketLocation` permission on the Amazon S3 bucket that the file is in. For more information, see Policy actions for Amazon S3 in the *Amazon Simple Storage Service User Guide*.

The following example policy adds the required permissions for retrieving a file from Amazon S3. Specify the name of your Amazon S3 bucket and configuration file name.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/folder_name/config_file_name"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ]
    }
  ]
}
```

Amazon ECS task IAM role

Your Amazon ECS tasks can have an IAM role associated with them. The permissions granted in the IAM role are vended to containers running in the task. This role allows your application code (running in the container) to use other AWS services. The task role is required when your application accesses other AWS services, such as Amazon S3.

> **Note**
>
> These permissions aren't accessed by the Amazon ECS container and Fargate agents. For the IAM permissions that Amazon ECS needs to pull container images and run the task, see [Amazon ECS task execution IAM role](#).

The following are the benefits of using task roles:

- **Separation of concerns**: If you're using the EC2 launch type, task IAM roles allow you to specify IAM permissions for your containers without requiring these permissions to be specified using EC2 instance profiles (for more information, see Using instance profiles in the *AWS Identity and Access Management User Guide*). Therefore, you can deploy your applications independently and uniformly on ECS container instances without needing to modify IAM permissions associated with EC2 instances.
- **Auditability**: Access and event logging are available through CloudTrail to ensure retrospective auditing. Task credentials have a context of '`taskArn`' that is attached to the session, so CloudTrail logs show which task the role credentials were vended for.
- **Uniform credentials delivery**: ECS delivers IAM role credentials to your containers and makes them accessible through a well-defined interface irrespective of the launch type associated with your tasks. On ECS Fargate, EC2 instance profiles are not available for containers in your tasks. Task IAM roles enable you to associate IAM permissions to your containers irrespective of the launch type when you use AWS SDK or AWS CLI in your containers. For more information about how the AWS SDK accesses these credentials, see Container credential provider.

Important

Containers are not a security boundary and the use of task IAM roles does not change this. Each task running on Fargate has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task. For EC2 and External Container Instances on ECS, there is no task isolation (unlike with Fargate) and containers can potentially access credentials for other tasks on the same container instance. They can also access permissions assigned to the ECS container instance role. Follow the recommendations in Roles recommendations to block access to the Amazon EC2 Instance Metadata Service for containers (For more information, see Use the Instance Metadata Service to access instance metadata in the *Amazon EC2 User Guide*).

Note that when you specify an IAM role for a task, the AWS CLI or other SDKs in the containers for that task use the AWS credentials provided by the task role exclusively and they do not inherit any IAM permissions from the Amazon EC2 or external instance they are running on.

Creating the task IAM role

When creating an IAM policy for your tasks to use, the policy must include the permissions that you want the containers in your tasks to assume. You can use an existing AWS managed policy, or you can create a custom policy from scratch that meets your specific needs. For more information, see Creating IAM policies in the *IAM User Guide*.

> **Important**
>
> For Amazon ECS tasks (for all launch types), we recommend that you use the IAM policy and role for your tasks. These credentials allow your task to make AWS API requests without calling `sts:AssumeRole` to assume the same role that is already associated with the task. If your task requires that a role assumes itself, you must create a trust policy that explicitly allows that role to assume itself. For more information, see Updating a role trust policy in the *IAM User Guide*.

After the IAM policy is created, you can create an IAM role which includes that policy which you reference in your Amazon ECS task definition. You can create the role using the **Elastic Container Service Task** use case in the IAM console. Then, you can attach your specific IAM policy to the role that gives the containers in your task the permissions you desire. The procedures below describe how to do this.

If you have multiple task definitions or services that require IAM permissions, you should consider creating a role for each specific task definition or service with the minimum required permissions for the tasks to operate so that you can minimize the access that you provide for each task.

For information about the service endpoint for your Region, see Service endpoints in the *Amazon Web Services General Reference Guide*.

The IAM task role must have a trust policy that specifies the `ecs-tasks.amazonaws.com` service. The `sts:AssumeRole` permission allows your tasks to assume an IAM role that's different from the one that the Amazon EC2 instance uses. This way, your task doesn't inherit the role associated with the Amazon EC2 instance. The following is an example trust policy. Replace the Region identifier and specify the AWS account number that you use when launching tasks.

> **Important**
>
> When creating your task IAM role, it is recommended that you use the `aws:SourceAccount` or `aws:SourceArn` condition keys in the trust relationship policy associated with the role to scope the permissions further to prevent the confused deputy security issue. Using the `aws:SourceArn` condition key to specify a specific cluster is not currently supported, you should use the wildcard to specify all clusters. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "ecs-tasks.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "ArnLike":{
            "aws:SourceArn":"arn:aws:ecs:us-west-2:111122223333:*"
            },
            "StringEquals":{
               "aws:SourceAccount":"111122223333"
            }
         }
      }
   ]
}
```

Use the following procedure to create a policy to retrieve objects from Amazon S3 with an example policy. Replace all *`user input`* with your own values.

AWS Management Console

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::my-task-secrets-bucket/*"
         ]
      }
   ]
}
```

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring](#) in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.

8. Choose **Create policy** to save your new policy.

AWS CLI

Replace all *user input* with your own values.

1. Create a file called `s3-policy.json` with the following content.

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::my-task-secrets-bucket/*"
         ]
      }
   ]
}
```

2. Use the following command to create the IAM policy using the JSON policy document file. Replace all *user input* with your own values.

```
aws iam create-policy \
      --policy-name taskRolePolicy \
      --policy-document file://s3-policy.json
```

Use the following procedure to create the service role.

AWS Management Console

To create the service role for Elastic Container Service (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **Elastic Container Service**, and then choose the **Elastic Container Service Task** use case.
5. Choose **Next**.
6. For **Add permissions**, search for and choose the policy you created.
7. Choose **Next**.
8. For **Role name**, enter a name for your role. For this example, type `AmazonECSTaskS3BucketRole` to name the role.
9. Review the role, and then choose **Create role**.

AWS CLI

1. Create a file named `ecs-tasks-trust-policy.json` that contains the trust policy to use for the task IAM role. The file should contain the following. Replace the Region identifier and specify the AWS account number that you use when launching tasks.

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "ecs-tasks.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole",
         "Condition":{
```

```
                "ArnLike":{
                "aws:SourceArn":"arn:aws:ecs:us-west-2:111122223333:*"
                },
                "StringEquals":{
                   "aws:SourceAccount":"111122223333"
                }
             }
          }
       ]
    }
```

2. Create an IAM role named `ecsTaskRole` using the trust policy created in the previous step.

```
aws iam create-role \
      --role-name ecsTaskRole \
      --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. Retrieve the ARN of the IAM policy you created using the following command. Replace *`taskRolePolicy`* with the name of the policy you created.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`taskRolePolicy`].Arn'
```

4. Attach the IAM policy you created to the `ecsTaskRole` role. Replace the `policy-arn` with the ARN of the policy that you created.

```
aws iam attach-role-policy \
      --role-name ecsTaskRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/taskRolePolicy
```

After you create the role, add additional permissions to the role for the following features.

Feature	Additional permissions
Use ECS Exec	ECS Exec permissions
Use an image from a private Amazon ECR repository	Amazon ECR permissions

Feature	Additional permissions
Use EC2 instances (Windows and Linux)	Amazon EC2 instances additional configura tion
Use external instances	External instance additional configuration
Use Windows EC2 instances	Amazon EC2 Windows instance additional configuration

Amazon ECR permissions

The following permissions are required when your application code needs to interact with Amazon ECR repositories directly. Note that for basic implementation where you only need to pull images from Amazon ECR, these permissions are not required at the task IAM role level. Instead, the Amazon ECS task execution role should have these permissions. For more information about the task execution role, see Amazon ECS task execution IAM role.

If your application code running in the container needs to interact with Amazon ECR APIs directly, you should add the following permissions to a task IAM role and include the task IAM role in your task definition. For more information, see Adding and Removing IAM Policies in the *IAM User Guide*.

Use the following policy for your task IAM role to add the required Amazon ECR permissions for container applications that need to interact with Amazon ECR directly:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

ECS Exec permissions

The ECS Exec feature requires a task IAM role to grant containers the permissions needed for communication between the managed SSM agent (`execute-command` agent) and the SSM service. You should add the following permissions to a task IAM role and include the task IAM role in your task definition. For more information, see Adding and Removing IAM Policies in the *IAM User Guide*.

Use the following policy for your task IAM role to add the required SSM permissions.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
       "Effect": "Allow",
       "Action": [
            "ssmmessages:CreateControlChannel",
            "ssmmessages:CreateDataChannel",
            "ssmmessages:OpenControlChannel",
            "ssmmessages:OpenDataChannel"
       ],
      "Resource": "*"
      }
   ]
}
```

Amazon EC2 instances additional configuration

We recommend that you limit the permissions in your container instance role to the minimal list of permissions used in the `AmazonEC2ContainerServiceforEC2Role` managed IAM policy.

Your Amazon EC2 instances require at least version `1.11.0` of the container agent to use task role; however, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS container agent. If you use an Amazon ECS-optimized AMI, your instance needs at least `1.11.0-1` of the `ecs-init` package. If your instances are using the latest Amazon ECS-optimized AMI, then they contain the required versions of the container agent and `ecs-init`. For more information, see Amazon ECS-optimized Linux AMIs.

If you are not using the Amazon ECS-optimized AMI for your container instances, add the `--net=host` option to your **docker run** command that starts the agent and the following agent configuration variables for your desired configuration (for more information, see Amazon ECS container agent configuration):

`ECS_ENABLE_TASK_IAM_ROLE=true`

Uses IAM roles for tasks for containers with the `bridge` and `default` network modes.

`ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true`

Uses IAM roles for tasks for containers with the `host` network mode. This variable is only supported on agent versions 1.12.0 and later.

For an example run command, see Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs). You will also need to set the following networking commands on your container instance so that the containers in your tasks can retrieve their AWS credentials:

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-destination 127.0.0.1:51679
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 51679
```

You must save these **iptables** rules on your container instance for them to survive a reboot. You can use the **iptables-save** and **iptables-restore** commands to save your **iptables** rules and restore them at boot. For more information, consult your specific operating system documentation.

To prevent containers run by tasks that use the `awsvpc` network mode from accessing the credential information supplied to the Amazon EC2 instance profile, while still allowing the permissions that are provided by the task role, set the `ECS_AWSVPC_BLOCK_IMDS` agent configuration variable to `true` in the agent configuration file and restart the agent. For more information, see Amazon ECS container agent configuration.

To prevent containers run by tasks that use the `bridge` network mode from accessing the credential information supplied to the Amazon EC2 instance profile, while still allowing the permissions that are provided by the task role, by running the following **iptables** command on your Amazon EC2 instances. This command doesn't affect containers in tasks that use the `host` or `awsvpc` network modes. For more information, see [Network mode](#).

- ```
  sudo yum install -y iptables-services; sudo iptables --insert DOCKER-USER 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
  ```

  You must save this **iptables** rule on your Amazon EC2 instance for it to survive a reboot. When using the Amazon ECS-optimized AMI, you can use the following command. For other operating systems, consult the documentation for that operating system.

  ```
  sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
  ```

External instance additional configuration

Your external instances require at least version `1.11.0` of the container agent to use task IAM roles; however, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent](#). If you are using an Amazon ECS-optimized AMI, your instance needs at least `1.11.0-1` of the `ecs-init` package. If your instances are using the latest Amazon ECS-optimized AMI, then they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized Linux AMIs](#).

If you are not using the Amazon ECS-optimized AMI for your container instances, add the `--net=host` option to your **docker run** command that starts the agent and the following agent configuration variables for your desired configuration (for more information, see [Amazon ECS container agent configuration](#)):

`ECS_ENABLE_TASK_IAM_ROLE=true`

Uses IAM roles for tasks for containers with the `bridge` and `default` network modes.

`ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true`

Uses IAM roles for tasks for containers with the `host` network mode. This variable is only supported on agent versions 1.12.0 and later.
  ```

For an example run command, see Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs). You will also need to set the following networking commands on your container instance so that the containers in your tasks can retrieve their AWS credentials:

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-destination 127.0.0.1:51679
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 51679
```

You must save these **iptables** rules on your container instance for them to survive a reboot. You can use the **iptables-save** and **iptables-restore** commands to save your **iptables** rules and restore them at boot. For more information, consult your specific operating system documentation.

Amazon EC2 Windows instance additional configuration

> **Important**
>
> This applies only to Windows containers on EC2 that use task roles.

The task role with Windows features requires additional configuration on EC2.

- When you launch your container instances, you must set the `-EnableTaskIAMRole` option in the container instances user data script. The `EnableTaskIAMRole` turns on the Task IAM roles feature for the tasks. For example:

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster 'windows' -EnableTaskIAMRole
</powershell>
```

- You must bootstrap your container with the networking commands that are provided in Amazon ECS container bootstrap script.
- You must create an IAM role and policy for your tasks. For more information, see Creating the task IAM role.
- The IAM roles for the task credential provider use port 80 on the container instance. Therefore, if you configure IAM roles for tasks on your container instance, your containers can't use port 80 for the host port in any port mappings. To expose your containers on port 80, we recommend

configuring a service for them that uses load balancing. You can use port 80 on the load balancer. By doing so, traffic can be routed to another host port on your container instances. For more information, see Use load balancing to distribute Amazon ECS service traffic.

- If your Windows instance is restarted, you must delete the proxy interface and initialize the Amazon ECS container agent again to bring the credential proxy back up.

Amazon ECS container bootstrap script

Before containers can access the credential proxy on the container instance to get credentials, the container must be bootstrapped with the required networking commands. The following code example script should be run on your containers when they start.

> **Note**
>
> You do not need to run this script when you use `awsvpc` network mode on Windows.

If you run Windows containers which include Powershell, then use the following script:

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

$gateway = (Get-NetRoute | Where { $_.DestinationPrefix -eq '0.0.0.0/0' } | Sort-Object
 RouteMetric | Select NextHop).NextHop
$ifIndex = (Get-NetAdapter -InterfaceDescription "Hyper-V Virtual Ethernet*" | Sort-
Object | Select ifIndex).ifIndex
New-NetRoute -DestinationPrefix 169.254.170.2/32 -InterfaceIndex $ifIndex -NextHop
 $gateway -PolicyStore ActiveStore # credentials API
New-NetRoute -DestinationPrefix 169.254.169.254/32 -InterfaceIndex $ifIndex -NextHop
 $gateway -PolicyStore ActiveStore # metadata API
```

If you run Windows containers that only have the Command shell, then use the following script:

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

for /f "tokens=1" %i  in ('netsh interface ipv4 show interfaces ^| findstr /x /r
 ".*vEthernet.*"') do set interface=%i
for /f "tokens=3" %i  in ('netsh interface ipv4 show addresses %interface% ^| findstr /
x /r ".*Default.Gateway.*"') do set gateway=%i
netsh interface ipv4 add route prefix=169.254.170.2/32 interface="%interface%"
 nexthop="%gateway%" store=active # credentials API
netsh interface ipv4 add route prefix=169.254.169.254/32 interface="%interface%"
 nexthop="%gateway%" store=active # metadata API
```

Amazon ECS container instance IAM role

Amazon ECS container instances, including both Amazon EC2 and external instances, run the Amazon ECS container agent and require an IAM role for the service to know that the agent belongs to you. Before you launch container instances and register them to a cluster, you must create an IAM role for your container instances to use. The role is created in the account that you use to log into the console or run the AWS CLI commands.

> **Important**
>
> If you are registering external instances to your cluster, the IAM role you use requires Systems Manager permissions as well. For more information, see Amazon ECS Anywhere IAM role.

Amazon ECS provides the `AmazonEC2ContainerServiceforEC2Role` managed IAM policy which contains the permissions needed to use the full Amazon ECS feature set. This managed policy can be attached to an IAM role and associated with your container instances. Alternatively, you can use the managed policy as a guide when creating a custom policy to use. The container instance role provides permissions needed for the Amazon ECS container agent and Docker daemon to call AWS APIs on your behalf. For more information on the managed policy, see AmazonEC2ContainerServiceforEC2Role.

Create the container instance role

> **Important**
>
> If you are registering external instances to your cluster, see Amazon ECS Anywhere IAM role.

You can manually create the role and attach the managed IAM policy for container instances to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. Use the following procedure to attach the managed IAM policy if needed.

AWS Management Console

To create the service role for Elastic Container Service (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **Elastic Container Service**, and then choose the **EC2 Role for Elastic Container Service** use case.
5. Choose **Next**.
6. In the **Permissions policies** section, verify that the **AmazonEC2ContainerServiceforEC2Role** policy is selected.

   > **Important**
   >
   > The **AmazonEC2ContainerServiceforEC2Role** managed policy should be attached to the container instance IAM role, otherwise you will receive an error using the AWS Management Console to create clusters.

7. Choose **Next**.
8. For **Role name**, enter **ecsInstanceRole**
9. Review the role, and then choose **Create role**.

AWS CLI

Replace all *user input* with your own values.

1. Create a file called `instance-role-trust-policy.json` with the following contents.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Use the following command to create the instance IAM role using the trust policy document.

```
aws iam create-role \
    --role-name ecsInstanceRole \
    --assume-role-policy-document file://instance-role-trust-policy.json
```

3. Create an instance profile named `ecsInstanceRole-profile` using the create-instance-profile command.

```
aws iam create-instance-profile --instance-profile-name ecsInstanceRole-profile
```

   Example response

```
{
    "InstanceProfile": {
        "InstanceProfileId": "AIPAJTLBPJLEGREXAMPLE",
```

```
        "Roles": [],
        "CreateDate": "2022-04-12T23:53:34.093Z",
        "InstanceProfileName": "ecsInstanceRole-profile",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:instance-profile/ecsInstanceRole-
profile"
    }
}
```

4. Add the *ecsInstanceRole* role to the *ecsInstanceRole-profile* instance profile.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name ecsInstanceRole-profile \
    --role-name ecsInstanceRole
```

5. Attach the `AmazonEC2ContainerServiceRoleForEC2Role` managed policy to the role using the following command.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role \
    --role-name ecsInstanceRole
```

After you create the role, add additional permissions to the role for the following features.

Feature	Additional permissions
Amazon ECR has the container image	Amazon ECR permissions
Have CloudWatch Logs monitor container instances	Monitoring container instances permissions
Host configuration files in an Amazon S3 bucket	Amazon S3 read-only access

Amazon ECR permissions

The Amazon ECS container instance role that you use with your container instances must have the following IAM policy permissions for Amazon ECR.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

If you use the `AmazonEC2ContainerServiceforEC2Role` managed policy for your container instances, then your role has the proper permissions. To check that your role supports Amazon ECR, see Amazon ECS Container Instance IAM Role in the *Amazon Elastic Container Service Developer Guide*.

Permissions required for setting the awsvpcTrunking account setting

Amazon ECS supports launching container instances with increased ENI density using supported Amazon EC2 instance types. When you use this feature, we recommend that you create 2 container instance roles. Enable the `awsvpcTrunking` account setting for one role and use that role for tasks that require ENI trunking. For information about the `awsvpcTrunking` account setting, see Access Amazon ECS features with account settings.

The container instance role that you use with your container instances must have the following IAM policy permissions for setting the account setting

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ListAccountSettings",
                "ecs:ListAttributes",
                "ecs:PutAccountSetting"
            ],
            "Resource": "*"
        }
    ]
}
```

In order to use the container instance role, add the following to instance user data:

```
#!/bin/bash
aws ecs put-account-setting --name awsvpcTrunking --value enabled --region region
ECS_CLUSTER=MyCluster>> /etc/ecs/ecs.config
EOF
```

For more information about adding user data to your EC2 instances, see [Run commands on your Linux instance at launch]() in the *Amazon EC2 User Guide*.

Amazon S3 read-only access

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance configuration at launch time. You can store a copy of your `ecs.config` file in a private bucket, use Amazon EC2 user data to install the AWS CLI and then copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

For more information about creating an `ecs.config` file, storing it in Amazon S3, and launching instances with this configuration, see [Storing Amazon ECS container instance configuration in Amazon S3]().

You can use the following AWS CLI command to allow Amazon S3 read-only access for your container instance role. Replace *`ecsInstanceRole`* with the name of the role that you created.

```
aws iam attach-role-policy \
      --role-name ecsInstanceRole \
      --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

You can also use the IAM console to add Amazon S3 read-only access (`AmazonS3ReadOnlyAccess`) to your role. For more information, see Updating permissions for a role in the *AWS Identity and Access Management User Guide*.

Permissions required for setting the awsvpcTrunking account setting

The Amazon ECS container instance role that you use with your container instances must have the following IAM policy permissions for setting the account setting

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:ListAccountSettings",
                "ecs:ListAttributes",
                "ecs:PutAccountSetting"
            ],
            "Resource": "*"
        }
    ]
}
```

Monitoring container instances permissions

Before your container instances can send log data to CloudWatch Logs, you must create an IAM policy to allow the Amazon ECS agent to write the customer's application logs to CloudWatch (normally handled through the `awslogs` driver). After you create the policy, attach that policy to `ecsInstanceRole`.

AWS Management Console

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": ["arn:aws:logs:*:*:*"]
        }
    ]
}
```

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring]() in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
8. Choose **Create policy** to save your new policy.

After you create the policy, attach the policy to the container instance role. For information about how to attach the policy to the role, see [Updating permissions for a role](#) in the *AWS Identity and Access Management User Guide*.

AWS CLI

1. Create a file called `instance-cw-logs.json` with the following content.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": ["arn:aws:logs:*:*:*"]
        }
    ]
}
```

2. Use the following command to create the IAM policy using the JSON policy document file.

```
aws iam create-policy \
      --policy-name cwlogspolicy \
      --policy-document file://instance-cw-logs.json
```

3. Retrieve the ARN of the IAM policy you created using the following command. Replace *cwlogspolicy* with the name of the policy you created.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`cwlogspolicy`].Arn'
```

4. Use the following command to attach the policy to the container instance IAM role using the policy ARN.

```
aws iam attach-role-policy \
```

```
        --role-name ecsInstanceRole \
        --policy-arn arn:aws:iam:111122223333:aws:policy/cwlogspolicy
```

Amazon ECS Anywhere IAM role

When you register an on-premises server or virtual machine (VM) to your cluster, the server or VM requires an IAM role to communicate with AWS APIs. You only need to create this IAM role once for each AWS account. However, this IAM role must be associated with each server or VM that you register to a cluster. This role is the `ECSAnywhereRole`. You can create this role manually. Alternatively, Amazon ECS can create the role on your behalf when you register an external instance in the AWS Management Console. You can use IAM console search to search for `ecsAnywhereRole` and see if your account already has the role. For more information, see IAM console search in the *IAM user guide*.

AWS provides two managed IAM policies that can be used when creating the ECS Anywhere IAM role, the `AmazonSSMManagedInstanceCore` and `AmazonEC2ContainerServiceforEC2Role` policies. The `AmazonEC2ContainerServiceforEC2Role` policy includes permissions that likely provide more access than you need. Therefore, depending on your specific use case, we recommend that you create a custom policy adding only the permissions from that policy that you require in it. For more information, see Amazon ECS container instance IAM role.

The task execution IAM role grants the Amazon ECS container agent permission to make AWS API calls on your behalf. When a task execution IAM role is used, it must be specified in your task definition. For more information, see Amazon ECS task execution IAM role.

The task execution role is required if any of the following conditions apply:

- You're sending container logs to CloudWatch Logs using the `awslogs` log driver.
- Your task definition specifies a container image that's hosted in an Amazon ECR private repository. However, if the `ECSAnywhereRole` role that's associated with your external instance also includes the permissions necessary to pull images from Amazon ECR then your task execution role doesn't need to include them.

Creating the Amazon ECS Anywhere role

Replace all *`user input`* with your own information.

1. Create a local file named `ssm-trust-policy.json` with the following trust policy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": [
      "ssm.amazonaws.com"
    ]},
    "Action": "sts:AssumeRole"
  }
}
```

2. Create the role and attach the trust policy by using the following AWS CLI command.

```
aws iam create-role --role-name ecsAnywhereRole --assume-role-policy-document
 file://ssm-trust-policy.json
```

3. Attach the AWS managed policies by using the following command.

```
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
 arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
```

You can also use the IAM custom trust policy workflow to create the role. For more information, see Creating a role using custom trust policies (console) in the *IAM User Guide*.

Amazon ECS infrastructure IAM role

An Amazon ECS infrastructure IAM role allows Amazon ECS to manage infrastructure resources in your clusters on your behalf, and is used when:

- You want to attach Amazon EBS volumes to your Fargate or EC2 launch type Amazon ECS tasks. The infrastructure role allows Amazon ECS to manage Amazon EBS volumes for your tasks.
- You want to use Transport Layer Security (TLS) to encrypt traffic between your Amazon ECS Service Connect services.
- You want to create Amazon VPC Lattice target groups.

When Amazon ECS assumes this role to take actions on your behalf, the events will be visible in AWS CloudTrail. If Amazon ECS uses the role to manage Amazon EBS volumes attached to your tasks, the CloudTrail log `roleSessionName` will be `ECSTaskVolumesForEBS`. If the role is used to encrypt traffic between your Service Connect services, the CloudTrail log `roleSessionName` will be `ECSServiceConnectForTLS`. If the role is used to create target groups for VPC Lattice, the CloudTrail log `roleSessionName` will be `ECSNetworkingWithVPCLattice`. You can use this name to search events in the CloudTrail console by filtering for **User name**.

Amazon ECS provides managed policies which contain the permissions required for volume attachment and TLS. For more information see, AmazonECSInfrastructureRolePolicyForVolumes, AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity, and AmazonECSInfrastructureRolePolicyForVpcLatticein the *AWS Managed Policy Reference Guide*.

Creating the Amazon ECS infrastructure role

Replace all *`user input`* with your own information.

1. Create a file named `ecs-infrastructure-trust-policy.json` that contains the trust policy to use for the IAM role. The file should contain the following:

   JSON

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "AllowAccessToECSForInfrastructureManagement",
         "Effect": "Allow",
         "Principal": {
           "Service": "ecs.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

2. Use the following AWS CLI command to create a role named `ecsInfrastructureRole` by using the trust policy that you created in the previous step.

   ```
   aws iam create-role \
   ```

```
      --role-name ecsInfrastructureRole \
      --assume-role-policy-document file://ecs-infrastructure-trust-policy.json
```

3. Depending on your use case, attach the AWS managed `AmazonECSInfrastructureRolePolicyForVolumes`, `AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity`, or `AmazonECSInfrastructureRolePolicyForVpcLattice` policy to the `ecsInfrastructureRole` role.

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSInfrastructureRolePolicyForVolumes
```

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity
```

You can also use the IAM console's **Custom trust policy** workflow to create the role. For more information, see [Creating a role using custom trust policies (console)] in the *IAM User Guide*.

Important

If the ECS infrastructure role is being used by Amazon ECS to manage Amazon EBS volumes attached to your tasks, ensure the following before you stop tasks that use Amazon EBS volumes.

- The role isn't deleted.
- The trust policy for the role isn't modified to remove Amazon ECS access (`ecs.amazonaws.com`).
- The managed policy `AmazonECSInfrastructureRolePolicyForVolumes` isn't removed. If you must modify the role's permissions, retain at least `ec2:DetachVolume`, `ec2:DeleteVolume`, and `ec2:DescribeVolumes` for volume deletion.

Deleting or modifying the role before stopping tasks with attached Amazon EBS volumes will result in the tasks getting stuck in `DEPROVISIONING` and the associated Amazon EBS volumes failing to delete. Amazon ECS will automatically retry at regular intervals to stop the task and delete the volume until the necessary permissions are restored. You can view a task's volume attachment status and associated status reason by using the DescribeTasks API.

After you create the file, you must grant your user permission to pass the role to Amazon ECS.

Permission to pass the infrastructure role to Amazon ECS

To use an ECS infrastructure IAM role, you must grant your user permission to pass the role to Amazon ECS. Attach the following `iam:PassRole` permission to your user. Replace *`ecsInfrastructureRole`* with the name of the infrastructure role that you created.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [

        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": ["arn:aws:iam::*:role/ecsInfrastructureRole"],
            "Condition": {
                "StringEquals": {"iam:PassedToService": "ecs.amazonaws.com"}
            }
        }
    ]
}
```

For more information about `iam:Passrole` and updating permissions for your user, see Granting a user permissions to pass a role to an AWS service and Changing permissions for an IAM user in the *AWS Identity and Access Management User Guide*.

Amazon ECS infrastructure IAM role for load balancers

An Amazon ECS infrastructure IAM role for load balancers allows Amazon ECS to manage load balancer resources in your clusters on your behalf, and is used when:

- You want to use blue/green deployments with Amazon ECS. The infrastructure role allows Amazon ECS to manage load balancer resources for your deployments.
- You need Amazon ECS to create, modify, or delete load balancer resources such as target groups and listeners during deployment operations.

When Amazon ECS assumes this role to take actions on your behalf, the events will be visible in AWS CloudTrail. If Amazon ECS uses the role to manage load balancer resources for your blue/green deployments, the CloudTrail log `roleSessionName` will be `ECSNetworkingWithELB` or `ecs-service-scheduler`. You can use this name to search events in the CloudTrail console by filtering for **User name**.

Amazon ECS provides a managed policy which contains the permissions required for load balancer management. For more information see, [AmazonECSInfrastructureRolePolicyForLoadBalancers]() in the *AWS Managed Policy Reference Guide*.

Creating the Amazon ECS infrastructure role for load balancers

Replace all *`user input`* with your own information.

1. Create a file named `ecs-infrastructure-trust-policy.json` that contains the trust policy to use for the IAM role. The file should contain the following:

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToECSForInfrastructureManagement",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Use the following AWS CLI command to create a role named `ecsInfrastructureRoleForLoadBalancers` by using the trust policy that you created in the previous step.

```
aws iam create-role \
      --role-name ecsInfrastructureRoleForLoadBalancers \
      --assume-role-policy-document file://ecs-infrastructure-trust-policy.json
```

3. Attach the AWS managed `AmazonECSInfrastructureRolePolicyForLoadBalancers` policy to the `ecsInfrastructureRoleForLoadBalancers` role.

```
aws iam attach-role-policy \
      --role-name ecsInfrastructureRoleForLoadBalancers \
      --policy-arn arn:aws:iam::aws:policy/
AmazonECSInfrastructureRolePolicyForLoadBalancers
```

You can also use the IAM console's **Custom trust policy** workflow to create the role. For more information, see Creating a role using custom trust policies (console) in the *IAM User Guide*.

Important

If the infrastructure role is being used by Amazon ECS to manage load balancer resources for your blue/green deployments, ensure the following before you delete or modify the role:

- The role isn't deleted while active deployments are in progress.
- The trust policy for the role isn't modified to remove Amazon ECS access (`ecs.amazonaws.com`).
- The managed policy `AmazonECSInfrastructureRolePolicyForLoadBalancers` isn't removed while active deployments are in progress.

Deleting or modifying the role during active blue/green deployments may result in deployment failures and could leave your services in an inconsistent state.

After you create the file, you must grant your user permission to pass the role to Amazon ECS.

Permission to pass the infrastructure role to Amazon ECS

To use an ECS infrastructure IAM role for load balancers, you must grant your user permission to pass the role to Amazon ECS. Attach the following `iam:PassRole` permission to your user. Replace *ecsInfrastructureRoleForLoadBalancers* with the name of the infrastructure role that you created.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource":
 ["arn:aws:iam::*:role/ecsInfrastructureRoleForLoadBalancers"],
            "Condition": {
                "StringEquals": {"iam:PassedToService": "ecs.amazonaws.com"}
            }
        }
    ]
}
```

For more information about `iam:Passrole` and updating permissions for your user, see [Granting a user permissions to pass a role to an AWS service]() and [Changing permissions for an IAM user]() in the *AWS Identity and Access Management User Guide*.

Amazon ECS CodeDeploy IAM Role

Before you can use the CodeDeploy blue/green deployment type with Amazon ECS, the CodeDeploy service needs permissions to update your Amazon ECS service on your behalf. These permissions are provided by the CodeDeploy IAM role (`ecsCodeDeployRole`).

> **Note**
>
> Users also require permissions to use CodeDeploy; these permissions are described in [Required IAM permissions]().

There are two managed policies provided. For more information, see one of the following in the *AWS Managed Policy Reference Guide*:

- AWSCodeDeployRoleForECS - gives CodeDeploy permission to update any resource using the associated action.
- AWSCodeDeployRoleForECSLimited - gives CodeDeploy more limited permissions.

Creating the CodeDeploy role

You can use the following procedures to create a CodeDeploy role for Amazon ECS

AWS Management Console

To create the service role for CodeDeploy (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **CodeDeploy**, and then choose the **CodeDeploy - ECS** use case.
5. Choose **Next**.
6. In the **Attach permissions policy** section, ensure that the **AWSCodeDeployRoleForECS** policy is selected.
7. Choose **Next**.
8. For **Role name**, enter **ecsCodeDeployRole**.
9. Review the role, and then choose **Create role**.

AWS CLI

Replace all *`user input`* with your own information.

1. Create a file named `codedeploy-trust-policy.json` that contains the trust policy to use for the CodeDeploy IAM role.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": ["codedeploy.amazonaws.com"]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Create an IAM role named `ecsCodedeployRole` using the trust policy created in the previous step.

```
aws iam create-role \
      --role-name ecsCodedeployRole \
      --assume-role-policy-document file://codedeploy-trust-policy.json
```

3. Attach the `AWSCodeDeployRoleForECS` or `AWSCodeDeployRoleForECSLimited` managed policy to the `ecsTaskRole` role.

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECS
```

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam::aws:policy/AWSCodeDeployRoleForECSLimited
```

When the tasks in your service need a task execution role, you must add the `iam:PassRole` permission for each task execution role or task role override to the CodeDeploy role as a policy.

Task execution role permissions

When the tasks in your service need a task execution role, you must add the `iam:PassRole` permission for each task execution role or task role override to the CodeDeploy role as a policy. For more information, see Amazon ECS task execution IAM role and Amazon ECS task IAM role. Then, you attach that policy to the CodeDeploy role

Create the policy

AWS Management Console

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::<aws_account_id>:role/
<ecsCodeDeployRole>"]
        }
    ]
}
```

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring](#) in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
8. Choose **Create policy** to save your new policy.

After you create the policy, attach the policy to the CodeDeploy role. For information about how to attach the policy to the role, see [Update permissions for a role](#) in the *AWS Identity and Access Management User Guide*.

AWS CLI

Replace all *`user input`* with your own information.

1. Create a file called `blue-green-iam-passrole.json` with the following content.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::*:role/code-deploy-role"],
            "Condition": {
                    "StringEquals": {"iam:PassedToService":
 "ecs.amazonaws.com"}
            }
        }
    ]
}
```

2. Use the following command to create the IAM policy using the JSON policy document file.

```
aws iam create-policy \
      --policy-name cdTaskExecutionPolicy \
      --policy-document file://blue-green-iam-passrole.json
```

3. Retrieve the ARN of the IAM policy you created using the following command.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`cdTaskExecutionPolicy`].Arn'
```

4. Use the following command to attach the policy to the CodeDeploy IAM role.

```
aws iam attach-role-policy \
      --role-name ecsCodedeployRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/cdTaskExecutionPolicy
```

Amazon ECS EventBridge IAM Role

Before you can use Amazon ECS scheduled tasks with EventBridge rules and targets, the EventBridge service needs permissions to run Amazon ECS tasks on your behalf. These permissions are provided by the EventBridge IAM role (`ecsEventsRole`).

The `AmazonEC2ContainerServiceEventsRole` policy is shown below.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["ecs:RunTask"],
            "Resource": ["*"]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["*"],
            "Condition": {
```

```
                "StringLike": {"iam:PassedToService": "ecs-tasks.amazonaws.com"}
            }
        },
        {
            "Effect": "Allow",
            "Action": "ecs:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ecs:CreateAction": ["RunTask"]
                }
            }
        }
    ]
}
```

If your scheduled tasks require the use of the task execution role, a task role, or a task role override, then you must add `iam:PassRole` permissions for each task execution role, task role, or task role override to the EventBridge IAM role. For more information about the task execution role, see [Amazon ECS task execution IAM role](#).

> **Note**
>
> Specify the full ARN of your task execution role or task role override.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
 "arn:aws:iam::111122223333:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
```

```
}
```

You can choose to let the AWS Management Console create the EventBridge role for you when you configure a scheduled task. For more information, see [Using Amazon EventBridge Scheduler to schedule Amazon ECS tasks](#).

Creating the EventBridge role

Replace all *user input* with your own information.

1. Create a file named `eventbridge-trust-policy.json` that contains the trust policy to use for the IAM role. The file should contain the following:

   JSON

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {
           "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

2. Use the following command to create an IAM role named `ecsEventsRole` by using the trust policy that you created in the previous step.

   ```
   aws iam create-role \
         --role-name ecsEventsRole \
         --assume-role-policy-document file://eventbridge-trust-policy.json
   ```

3. Attach the AWS managed `AmazonEC2ContainerServiceEventsRole` to the `ecsEventsRole` role using the following command.

   ```
   aws iam attach-role-policy \
   ```

```
      --role-name ecsEventsRole \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceEventsRole
```

You can also use the IAM console's **Custom trust policy** workflow (https://console.aws.amazon.com/iam/) to create the role. For more information, see Creating a role using custom trust policies (console) in the *IAM User Guide*.

Attaching a policy to the `ecsEventsRole` role

You can use the following procedures to add permissions for the task execution role to the EventBridge IAM role.

AWS Management Console

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": ["arn:aws:iam::<aws_account_id>:role/
<ecsTaskExecutionRole_or_TaskRole_name>"]
        }
    ]
}
```

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring]() in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.

8. Choose **Create policy** to save your new policy.

After you create the policy, attach the policy to the EventBridge role. For information about how to attach the policy to the role, see [Update permissions for a role]() in the *AWS Identity and Access Management User Guide*.

AWS CLI

Replace all *user input* with your own information.

1. Create a file called `ev-iam-passrole.json` with the following content.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
 "arn:aws:iam::111122223333:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

2. Use the following AWS CLI command to create the IAM policy using the JSON policy document file.

```
aws iam create-policy \
      --policy-name eventsTaskExecutionPolicy \
      --policy-document file://ev-iam-passrole.json
```

3. Retrieve the ARN of the IAM policy you created using the following command.

```
aws iam list-policies --scope Local --query 'Policies[?
PolicyName==`eventsTaskExecutionPolicy`].Arn'
```

4. Use the following command to attach the policy to the EventBridge IAM role by using the policy ARN.

```
aws iam attach-role-policy \
      --role-name ecsEventsRole \
      --policy-arn arn:aws:iam:111122223333:aws:policy/eventsTaskExecutionPolicy
```

Permissions required for the Amazon ECS console

Following the best practice of granting least privilege, you can use the `AmazonECS_FullAccess` managed policy as a template for creating you own custom policy. That way, you can take away or add permissions to and from the managed policy based on your specific requirements. For more information, see AmazonECS_FullAccess in the *AWS Managed Policy Reference*.

Permissions for creating IAM roles

The following actions require additional permissions in order to complete the operation:

- Registering an external instance - for more information, see Amazon ECS Anywhere IAM role
- Registering a task definition - for more information, see Amazon ECS task execution IAM role
- Creating an EventBridge rule to use for scheduling tasks - for more information, see Amazon ECS EventBridge IAM Role

You can add these permissions by creating a role in IAM before you use them in the Amazon ECS console. If you do not create the roles, the Amazon ECS console creates then on your behalf.

Permissions required for registering an external instance to a cluster

You need additional permissions when you register an external instance to a cluster and you want to create a new external instance (`ecsExternalInstanceRole`) role.

The following additional permissions are required:

- `iam`– Allows principals to create and list IAM roles and their attached policies.
- `ssm` – Allows principals to register the external instance with Systems Manager.

> **Note**
>
> In order to choose an existing `ecsExternalInstanceRole`, you must have the `iam:GetRole` and `iam:PassRole` permissions.

The following policy contains the required permissions, and limits the actions to the `ecsExternalInstanceRole` role.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:CreateInstanceProfile",
              "iam:AddRoleToInstanceProfile",
              "iam:ListInstanceProfilesForRole",
              "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsExternalInstanceRole"
        },
        {
            "Effect": "Allow",
            "Action": ["iam:PassRole","ssm:CreateActivation"],
            "Resource": "arn:aws:iam::*:role/ecsExternalInstanceRole"
        }
    ]
}
```

Permissions required for registering a task definition

You need additional permissions when you register a task definition and you want to create a new task execution (`ecsTaskExecutionRole`) role.

The following additional permissions are required:

- `iam`– Allows principals to create and list IAM roles and their attached policies.

> **Note**
>
> In order to choose an existing `ecsTaskExecutionRole`, you must have the `iam:GetRole` permission.

The following policy contains the required permissions, and limits the actions to the `ecsTaskExecutionRole` role.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsTaskExecutionRole"
        }
    ]
}
```

Permissions required for creating an EventBridge rule for scheduled tasks

You need additional permissions when you schedule a task and you want to create a new CloudWatch Events role (`ecsEventsRole`) role.

The following additional permissions are required:

- `iam`– Allows principals to create and list IAM roles and their attached policies, and to allow Amazon ECS to pass the role to other services to assume the role.

> **Note**
>
> In order to choose an existing `ecsEventsRole`, you must have the `iam:GetRole` and `iam:PassRole` permissions.

The following policy contains the required permissions, and limits the actions to the `ecsEventsRole` role.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:GetRole",
              "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsEventsRole"
        }
    ]
}
```

Permissions required for viewing service deployments

When you follow the best practice of granting least privilege, you need to add additional permissions in order to view service deployments in the console.

You need access to the following actions:

- ListServiceDeployments
- DescribeServiceDeployments
- DescribeServiceRevisions

You need access to the following resources:

- Service
- Service deployment

- Service revision

The following example policy contains the required permissions, and limits the actions to a specified service.

Replace the `account`, `cluster-name`, and `service-name` with your values.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "ecs:ListServiceDeployments",
              "ecs:DescribeServiceDeployments",
              "ecs:DescribeServiceRevisions"
            ],
            "Resource": [
             "arn:aws:ecs:us-east-1:123456789012:service/cluster-name/service-name",
             "arn:aws:ecs:us-east-1:123456789012:service-deployment/cluster-name/
service-name/*",
             "arn:aws:ecs:us-east-1:123456789012:service-revision/cluster-name/service-
name/*"
            ]
        }
    ]
}
```

Permissions required to view Amazon ECS lifecycle events in Container Insights

The following permissions are required to view the lifecycle events. Add the following permissions as an inline policy to the role. For more information, see [Adding and Removing IAM Policies](#).

- events:DescribeRule
- events:ListTargetsByRule
- logs:DescribeLogGroups

JSON

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule",
        "events:ListTargetsByRule",
        "logs:DescribeLogGroups"
      ],
      "Resource": "*"
    }
  ]
}
```

Permissions required for enabling Amazon ECS lifecycle events inContainer Insights

The following permissions are required to configure the lifecycle events:

- events:PutRule
- events:PutTargets
- logs:CreateLogGroup

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "logs:CreateLogGroup"
      ],
      "Resource": "*"
    }
  ]
}
```

Permissions required for the Amazon ECS console with AWS CloudFormation

Before using the AWS Management Console to create your resources, you'll need to make sure to have the correct IAM permissions. For information on how to set up permissions for the Amazon ECS console in general first, see Permissions required for the Amazon ECS console.

The Amazon ECS console is powered by AWS CloudFormation and requires additional IAM permissions in the following cases:

- Creating a cluster
- Creating a service
- Creating a capacity provider

You can create a policy for the additional permissions, and then attach them to the IAM role you use to access the console. For more information, see Creating IAM policies in the *IAM User Guide*.

Permissions required for creating a cluster

When you create a cluster in the console, you need additional permissions that grant you permissions to manage AWS CloudFormation stacks.

The following additional permissions are required:

- `cloudformation` – Allows principals to create and manage AWS CloudFormation stacks. This is required when creating Amazon ECS clusters using the AWS Management Console and the subsequent managing of those clusters.
- `ssm` – Allows AWS CloudFormation to reference latest Amazon ECS-optimized AMI. This is required when creating Amazon ECS clusters using the AWS Management Console.

The following policy contains the required AWS CloudFormation permissions, and limits the actions to resources created in the Amazon ECS console.

```
{
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
                "cloudformation:CreateStack",
```

```
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack"
             ],
            "Resource": [
                "arn:*:cloudformation:*:*:stack/Infra-ECS-Cluster-*"
            ]
      },
      {
          "Effect": "Allow",
          "Action": "ssm:GetParameters",
          "Resource": [
            "arn:aws:ssm:*:*:parameter/aws/service/ecs/optimized-ami/amazon-linux-2*/
*",
            "arn:aws:ssm:*:*:parameter/aws/service/ecs/optimized-ami/amazon-
linux-2023*/*"
          ]
      }
   ]
}
```

If you have not created the Amazon ECS container instance role (`ecsInstanceRole`), and you are creating a cluster that uses Amazon EC2 instances, then the console will create the role on your behalf.

In addition, if you use Auto Scaling groups, then you need additional permissions so that the console can add tags to the auto scaling groups when using the cluster auto scaling feature.

The following additional permissions are required:

- `autoscaling` – Allows the console to tag Amazon EC2 Auto Scaling group. This is required when managing Amazon EC2 auto scaling groups when using the cluster auto scaling feature. The tag is the ECS-managed tag that the console automatically adds to the group to indicate is was created in the console.
- `iam`– Allows principals to list IAM roles and their attached policies. Principals can also list instance profiles available to your Amazon EC2 instances.

The following policy contains the required IAM permissions, and limits the actions to the `ecsInstanceRole` role.

The Auto Scaling permissions are not limited.

```
{
  "Statement": [
      {
          "Effect": "Allow",
            "Action": [
              "iam:AttachRolePolicy",
              "iam:CreateRole",
              "iam:CreateInstanceProfile",
              "iam:AddRoleToInstanceProfile",
              "iam:ListInstanceProfilesForRole",
              "iam:GetRole"
            ],
            "Resource": "arn:aws:iam::*:role/ecsInstanceRole"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:CreateOrUpdateTags",
            "Resource": "*"
        }
    ]
}
```

Permissions required for creating a service

When you create a service in the console, you need additional permissions that grant you permissions to manage AWS CloudFormation stacks. The following additional permissions are required:

- `cloudformation` – Allows principals to create and manage AWS CloudFormation stacks. This is required when creating Amazon ECS services using the AWS Management Console and the subsequent managing of those services.

The following policy contains the required permissions, and limits the actions to resources created in the Amazon ECS console.

```
{
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
                "cloudformation:CreateStack",
```

```
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack"
             ],
            "Resource": [
                "arn:*:cloudformation:*:*:stack/ECS-Console-V2-Service-*"
            ]
        }
    ]
}
```

Permissions required for Lambda functions in Amazon ECS blue/green deployments

When you use Lambda functions as deployment lifecycle hooks in Amazon ECS blue/green deployments, you need to create an IAM role with specific permissions. This role allows Amazon ECS to invoke your Lambda functions at various stages of the deployment lifecycle.

The following additional permissions are required:

- `lambda:InvokeFunction` – Allows Amazon ECS to invoke Lambda functions configured as lifecycle hooks during the deployment process.

The following policy contains the required permissions for Lambda lifecycle hooks:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:*:*:function:*"
        }
    ]
}
```

For the trust policy, you need to allow the service to assume this role:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ecs.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

IAM permissions required for Amazon ECS service auto scaling

Service Auto Scaling is made possible by a combination of the Amazon ECS, CloudWatch, and Application Auto Scaling APIs. Services are created and updated with Amazon ECS, alarms are created with CloudWatch, and scaling policies are created with Application Auto Scaling.

In addition to the standard IAM permissions for creating and updating services, the following permissions are required to interact with Service Auto Scaling settings as shown in the following example policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:*",
                "ecs:DescribeServices",
                "ecs:UpdateService",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm",
```

```
                "cloudwatch:DeleteAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:DescribeAlarmsForMetric",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "cloudwatch:DisableAlarmActions",
                "cloudwatch:EnableAlarmActions",
                "iam:CreateServiceLinkedRole",
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:Get*",
                "sns:List*"
            ],
            "Resource": ["*"]
        }
    ]
}
```

The Create Amazon ECS service example and Update Amazon ECS service example IAM policy examples show the required permissions to use Service Auto Scaling in the AWS Management Console.

The Application Auto Scaling service also needs permission to describe your Amazon ECS services and CloudWatch alarms, and permissions to modify your service's desired count on your behalf. The `sns:` permissions are for the notifications that CloudWatch sends to an Amazon SNS topic when a threshold has been exceeded. If you use automatic scaling for your Amazon ECS services, it creates a service-linked role named `AWSServiceRoleForApplicationAutoScaling_ECSService`. This service-linked role grants Application Auto Scaling permission to describe the alarms for your policies, to monitor the current running task count of the service, and to modify the desired count of the service. The original managed Amazon ECS role for Application Auto Scaling was `ecsAutoscaleRole`, but it is no longer required. The service-linked role is the default role for Application Auto Scaling. For more information, see Service-linked roles for Application Auto Scaling in the *Application Auto Scaling User Guide*.

If you created your Amazon ECS container instance role before CloudWatch metrics are available for Amazon ECS, you might need to add the `ecs:StartTelemetrySession` permission. For more information, see Considerations.

Grant permission to tag resources on creation

The following tag-on create Amazon ECS API actions allow you to specify tags when you create the resource. If tags are specified in the resource-creating action, AWS performs additional authorization to verify that the correct permissions are assigned to create tags.

- `CreateCapacityProvider`
- `CreateCluster`
- `CreateService`
- `CreateTaskSet`
- `RegisterContainerInstance`
- `RegisterTaskDefinition`
- `RunTask`
- `StartTask`

You can use resource tags to implement attribute-based control (ABAC). For more information, see the section called “Control access to Amazon ECS resources using resource tags” and *Tagging resources*.

To allow tagging on creation, create or modify a policy to include both the permissions to use the action that creates the resource, such as `ecs:CreateCluster` or `ecs:RunTask` and the `ecs:TagResource` action.

The following example demonstrates a policy that allows users to create clusters and add tags during the cluster creation. Users are not permitted to tag any existing resources (they cannot call the `ecs:TagResource` action directly).

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "ecs:CreateCluster"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
```

```
      "Action": [
         "ecs:TagResource"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
                  "ecs:CreateAction": [
                      "CreateCluster",
                      "CreateCapacityProvider",
                      "CreateService",
                      "CreateTaskSet",
                      "RegisterContainerInstance",
                      "RegisterTaskDefinition",
                      "RunTask",
                      "StartTask"
                  ]
          }
       }
    }
  ]
}
```

The `ecs:TagResource` action is only evaluated if tags are applied during the resource-creating action. Therefore, a user that has permissions to create a resource (assuming there are no tagging conditions) does not require permissions to use the `ecs:TagResource` action if no tags are specified in the request. However, if the user attempts to create a resource with tags, the request fails if the user does not have permissions to use the `ecs:TagResource` action.

Amazon ECS control access to specific tags

You can use additional conditions in the `Condition` element of your IAM policies to control the tag keys and values that can be applied to resources.

The following condition keys can be used with the examples in the preceding section:

- `aws:RequestTag`: To indicate that a particular tag key or tag key and value must be present in a request. Other tags can also be specified in the request.
  - Use with the `StringEquals` condition operator to enforce a specific tag key and value combination, for example, to enforce the tag `cost-center=cc123`:

    ```
    "StringEquals": { "aws:RequestTag/cost-center": "cc123" }
    ```

  - Use with the `StringLike` condition operator to enforce a specific tag key in the request; for example, to enforce the tag key `purpose`:

    ```
    "StringLike": { "aws:RequestTag/purpose": "*" }
    ```

- `aws:TagKeys`: To enforce the tag keys that are used in the request.
  - Use with the `ForAllValues` modifier to enforce specific tag keys if they are provided in the request (if tags are specified in the request, only specific tag keys are allowed; no other tags are allowed). For example, the tag keys `environment` or `cost-center` are allowed:

    ```
    "ForAllValues:StringEquals": { "aws:TagKeys": ["environment","cost-center"] }
    ```

  - Use with the `ForAnyValue` modifier to enforce the presence of at least one of the specified tag keys in the request. For example, at least one of the tag keys `environment` or `webserver` must be present in the request:

    ```
    "ForAnyValue:StringEquals": { "aws:TagKeys": ["environment","webserver"] }
    ```

These condition keys can be applied to resource-creating actions that support tagging, as well as the `ecs:TagResource` action. To learn whether an Amazon ECS API action supports tagging, see Actions, resources, and condition keys for Amazon ECS.

To force users to specify tags when they create a resource, you must use the `aws:RequestTag` condition key or the `aws:TagKeys` condition key with the `ForAnyValue` modifier on the resource-creating action. The `ecs:TagResource` action is not evaluated if a user does not specify tags for the resource-creating action.

For conditions, the condition key is not case-sensitive and the condition value is case-sensitive. Therefore, to enforce the case-sensitivity of a tag key, use the `aws:TagKeys` condition key, where the tag key is specified as a value in the condition.

For more information about multi-value conditions, see Conditions with multiple context keys or values in the *IAM User Guide*.

Control access to Amazon ECS resources using resource tags

When you create an IAM policy that grants users permission to use Amazon ECS resources, you can include tag information in the `Condition` element of the policy to control access based on tags.

This is known as attribute-based access control (ABAC). ABAC provides better control over which resources a user can modify, use, or delete. For more information, see What is ABAC for AWS?

For example, you can create a policy that allows users to delete a cluster, but denies the action if the cluster has the tag `environment=production`. To do this, you use the `aws:ResourceTag` condition key to allow or deny access to the resource based on the tags that are attached to the resource.

```
"StringEquals": { "aws:ResourceTag/environment": "production" }
```

To learn whether an Amazon ECS API action supports controlling access using the `aws:ResourceTag` condition key, see Actions, resources, and condition keys for Amazon ECS. Note that the `Describe` actions do not support resource-level permissions, so you must specify them in a separate statement without conditions.

For example IAM policies, see Amazon ECS Example policies .

If you allow or deny users access to resources based on tags, you must consider explicitly denying users the ability to add those tags to or remove them from the same resources. Otherwise, it's possible for a user to circumvent your restrictions and gain access to a resource by modifying its tags.

Amazon ECS Example policies

You can use IAM policies to grant users permissions to view and work with specific resources in the Amazon ECS console. You can use the example policies in the previous section; however, they are designed for requests that are made with the AWS CLI or an AWS SDK.

Example: Allow users to delete an Amazon ECS cluster based on tags

The following policy allows users to delete clusters when the tag has a key/value pair of "Purpose/Testing".

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
```

```
                "ecs:DeleteCluster"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:ecs:us-east-1:111122223333:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Purpose": "Testing"
                }
            }
        }
    ]
}
```

Troubleshooting Amazon Elastic Container Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon ECS and IAM.

Topics

I am not authorized to perform an action in Amazon ECS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `ecs:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 ecs:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `ecs:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon ECS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon ECS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon ECS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon ECS supports these features, see How Amazon Elastic Container Service works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.

- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Additional troubleshooting resources

The following pages provide information about error codes:

- Amazon ECS stopped tasks error messages
- Viewing Amazon ECS service event messages

IAM best practices for Amazon ECS

You can use AWS Identity and Access Management (IAM) to manage and control access to your AWS services and resources through rule-based policies for authentication and authorization purposes. More specifically, through this service, you control access to your AWS resources by using policies that are applied to users, groups, or roles. Among these three, users are accounts that can have access to your resources. And, an IAM role is a set of permissions that can be assumed by an authenticated identity, which isn't associated with a particular identity outside of IAM. For more information, see Amazon ECS overview of access management: Permissions and policies.

Follow the policy of least privileged access

Create policies that are scoped to allow users to perform their prescribed jobs. For example, if a developer needs to periodically stop a task, create a policy that only permits that particular action. The following example only allows a user to stop a task that belongs to a particular `task_family` on a cluster with a specific Amazon Resource Name (ARN). Referring to an ARN in a condition is also an example of using resource-level permissions. You can use resource-level permissions to specify the resource that you want an action to apply to.

JSON

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
```

```
                    "Effect": "Allow",
                    "Action": [
                       "ecs:StopTask"
                    ],
                   "Resource": "arn:aws:ecs:us-east-1:123456789012:task/my-cluster/
*",
                   "Condition": {
                           "ArnEquals": {
                                   "ecs:cluster": "arn:aws:ecs:us-
east-1:123456789012:cluster/my-cluster"
                           }
                   }
               }
           ]
}
```

Have cluster resources serve as the administrative boundary

Policies that are too narrowly scoped can cause a proliferation of roles and increase administrative overhead. Rather than creating roles that are scoped to particular tasks or services only, create roles that are scoped to clusters and use the cluster as your primary administrative boundary.

Create automated pipelines to isolate end-users from the API

You can limit the actions that users can use by creating pipelines that automatically package and deploy applications onto Amazon ECS clusters. This effectively delegates the job of creating, updating, and deleting tasks to the pipeline. For more information, see Tutorial: Amazon ECS standard deployment with CodePipeline in the *AWS CodePipeline User Guide*.

Use policy conditions for an added layer of security

When you need an added layer of security, add a condition to your policy. This can be useful if you're performing a privileged operation or when you need to restrict the set of actions that can be performed against particular resources. The following example policy requires multi-factor authorization when deleting a cluster.

JSON

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DeleteCluster"
      ],
      "Condition": {
        "Bool": {
          "aws:MultiFactorAuthPresent": "true"
        }
      },
    "Resource": ["*"]
    }
  ]
}
```

Tags applied to services are propagated to all the tasks that are part of that service. Because of this, you can create roles that are scoped to Amazon ECS resources with specific tags. In the following policy, an IAM principal starts and stops all tasks with a tag-key of `Department` and a tag-value of `Accounting`.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:RunTask"
            ],
            "Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Department": "Accounting"}
            }
        }
    ]
}
```

Periodically audit access to the APIs

A user might change roles. After they change roles, the permissions that were previously granted to them might no longer apply. Make sure that you audit who has access to the Amazon ECS APIs and whether that access is still warranted. Consider integrating IAM with a user lifecycle management solution that automatically revokes access when a user leaves the organization. For more information, see AWS security audit guidelines in the *AWS Identity and Access Management User Guide*.

Logging and Monitoring in Amazon Elastic Container Service

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Elastic Container Service and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon ECS resources and responding to potential incidents:

Amazon CloudWatch Alarms

Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Monitor Amazon ECS using CloudWatch.

For services with tasks that use the Fargate launch type, you can use CloudWatch alarms to scale in and scale out the tasks in your service based on CloudWatch metrics, such as CPU and memory utilization. For more information, see Automatically scale your Amazon ECS service.

For clusters with tasks or services using the EC2 launch type, you can use CloudWatch alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation.

Amazon CloudWatch Logs

Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the `awslogs` log driver in your task definitions. For more information, see Using the awslogs driver.

You can also monitor, store, and access the operating system and Amazon ECS container agent log files from your Amazon ECS container instances. This method for accessing logs can be used for containers using the EC2 launch type..

Amazon CloudWatch Events

Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see Automate responses to Amazon ECS errors using EventBridge in this guide and EventBridge is the evolution of Amazon CloudWatch Events in the *Amazon EventBridge User Guide*.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon ECS. Using the information collected by CloudTrail, you can determine the request that was made to Amazon ECS, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Log Amazon ECS API calls using AWS CloudTrail.

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

For more information, see AWS Trusted Advisor in the *Support User Guide*.

AWS Compute Optimizer

AWS Compute Optimizer is a service that analyzes the configuration and utilization metrics of your AWS resources. It reports whether your resources are optimal, and generates optimization recommendations to reduce the cost and improve the performance of your workloads.

For more information, see AWS Compute Optimizer recommendations for Amazon ECS.

Another important part of monitoring Amazon ECS involves manually monitoring those items that the CloudWatch alarms don't cover. The CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your container instances and the containers in your tasks.

Compliance validation for Amazon Elastic Container Service

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.

- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Compliance and security best practices for Amazon ECS

Your compliance responsibility when using Amazon ECS is determined by the sensitivity of your data, the compliance objectives of your company, and applicable laws and regulations.

Payment Card Industry Data Security Standards (PCI DSS)

It's important that you understand the complete flow of cardholder data (CHD) within the environment when adhering to PCI DSS. The CHD flow determines the applicability of the PCI DSS, defines the boundaries and components of a cardholder data environment (CDE), and therefore the scope of a PCI DSS assessment. Accurate determination of the PCI DSS scope is key to defining the security posture and ultimately a successful assessment. Customers must have a procedure for scope determination that assures its completeness and detects changes or deviations from the scope.

The temporary nature of containerized applications provides additional complexities when auditing configurations. As a result, customers need to maintain an awareness of all container configuration parameters to ensure compliance requirements are addressed throughout all phases of a container lifecycle.

For additional information on achieving PCI DSS compliance on Amazon ECS, refer to the following whitepapers.

- Architecting on Amazon ECS for PCI DSS compliance

HIPAA (U.S. Health Insurance Portability and Accountability Act)

Using Amazon ECS with workloads that process protected health information (PHI) requires no additional configuration. Amazon ECS acts as an orchestration service that coordinates the launch of containers on Amazon EC2. It doesn't operate with or upon data within the workload being orchestrated. Consistent with HIPAA regulations and the AWS Business Associate Addendum, PHI should be encrypted in transit and at-rest when accessed by containers launched with Amazon ECS.

Various mechanisms for encrypting at-rest are available with each AWS storage option, such as Amazon S3, Amazon EBS, and AWS KMS. You can deploy an overlay network (such as VNS3 or

Weave Net) to ensure complete encryption of PHI transferred between containers or to provide a redundant layer of encryption. You should also use complete logging, and direct all container logs to Amazon CloudWatch. For information about using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

AWS Security Hub

Use AWS Security Hub. This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.

Amazon GuardDuty with Amazon ECS Runtime Monitoring

Amazon GuardDuty is a threat detection service that helps protect your accounts, containers, workloads, and the data within your AWS environment. Using machine learning (ML) models, and anomaly and threat detection capabilities, GuardDuty continuously monitors different log sources and runtime activity to identify and prioritize potential security risks and malicious activities in your environment.

Use Runtime Monitoring in GuardDuty to identify malicious or unauthorized behavior. Runtime Monitoring protects workloads running on Fargate and EC2 by continuously monitoring AWS log and networking activity to identify malicious or unauthorized behavior. Runtime Monitoring uses a lightweight, fully managed GuardDuty security agent that analyzes on-host behavior, such as file access, process execution, and network connections. This covers issues including escalation of privileges, use of exposed credentials, or communication with malicious IP addresses, domains, and the presence of malware on your Amazon EC2 instances and container workloads. For more information, see GuardDuty Runtime Monitoring in the *GuardDuty User Guide*.

Compliance recommendations

We recommend that you engage the compliance program owners within your business early and use the AWS shared responsibility model to identify compliance control ownership for success with the relevant compliance programs. For more information, see AWS shared responsibility model for Amazon ECS .

AWS Fargate Federal Information Processing Standard (FIPS-140)

Federal Information Processing Standard (FIPS). FIPS-140 is a U.S. and Canadian government standard that specifies the security requirements for cryptographic modules that protect sensitive information. FIPS-140 defines a set of validated cryptography functions that can be used to encrypt data in transit and data at rest.

When you turn on FIPS-140 compliance, you can run workloads on Fargate in a manner that is compliant with FIPS-140. For more information about FIPS-140 compliance, see Federal Information Processing Standard (FIPS) 140-3.

AWS Fargate FIPS-140 Considerations

Consider the following when using FIPS-140 compliance on Fargate:

- FIPS-140 compliance is only available in the AWS GovCloud (US) Regions.
- Fargate supports FIPS-140 version 140.3
- FIPS-140 compliance is turned off by default. You must turn it on.
- Your tasks must use the following configuration for FIPS-140 compliance:
  - The `operatingSystemFamily` must be `LINUX`.
  - The `cpuArchitecture` must be `X86_64`.
  - The Fargate platform version must be `1.4.0` or later.

Use FIPS on Fargate

Use the following procedure to use FIPS-140 compliance on Fargate.

1. Turn on FIPS-140 compliance. For more information, see the section called “AWS Fargate Federal Information Processing Standard (FIPS-140) compliance”.
2. You can optionally use ECS Exec to run the following command to verify the FIPS-140 compliance status for a cluster.

   Replace *`cluster-name`* with the name of your cluster, *`task-id`* with the ID or ARN of your task, and *`container-name`* with the name of the container in your task you want to run the command against.

A return value of "1" indicates that you are using FIPS.

```
aws ecs execute-command \
    --cluster cluster-name \
    --task task-id \
    --container container-name \
    --interactive \
    --command "cat /proc/sys/crypto/fips_enabled"
```

Use CloudTrail for Fargate FIPS-140 auditing

CloudTrail is turned on in your AWS account when you create the account. When API and console activity occurs in Amazon ECS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon ECS, create a trail which CloudTrail uses to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the section called “Log Amazon ECS API calls using AWS CloudTrail”.

The following example shows a CloudTrail log entry that demonstrates the `PutAccountSettingDefault` API action:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAIV5AJI5LXF5EXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/jdoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIPWIOFC3EXAMPLE",
    },
    "eventTime": "2023-03-01T21:45:18Z",
    "eventSource": "ecs.amazonaws.com",
    "eventName": "PutAccountSettingDefault",
```

```
    "awsRegion": "us-gov-east-1",
    "sourceIPAddress": "52.94.133.131",
    "userAgent": "aws-cli/2.9.8 Python/3.9.11 Windows/10 exe/AMD64 prompt/off command/
ecs.put-account-setting",
    "requestParameters": {
        "name": "fargateFIPSMode",
        "value": "enabled"
    },
    "responseElements": {
        "setting": {
            "name": "fargateFIPSMode",
            "value": "enabled",
            "principalArn": "arn:aws:iam::123456789012:user/jdoe"
        }
    },
    "requestID": "acdc731e-e506-447c-965d-f5f75EXAMPLE",
    "eventID": "6afced68-75cd-4d44-8076-0beEXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "ecs-fips.us-gov-east-1.amazonaws.com"
    }
}
```

Infrastructure Security in Amazon Elastic Container Service

As a managed service, Amazon Elastic Container Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access Amazon ECS through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.

- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location. Amazon ECS supports resource-based access policies, which can include restrictions based on the source IP address, so make sure that the policies account for the IP address for the network location. You can also use Amazon ECS policies to control access from specific Amazon Virtual Private Cloud endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon ECS resource from only the specific VPC within the AWS network. For more information, see Amazon ECS interface VPC endpoints (AWS PrivateLink).

Amazon ECS interface VPC endpoints (AWS PrivateLink)

You can improve the security posture of your VPC by configuring Amazon ECS to use an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that allows you to privately access Amazon ECS APIs by using private IP addresses. AWS PrivateLink restricts all network traffic between your VPC and Amazon ECS to the Amazon network. You don't need an internet gateway, a NAT device, or a virtual private gateway.

For more information about AWS PrivateLink and VPC endpoints, see VPC endpoints in the *Amazon VPC User Guide*.

Considerations

Considerations for endpoints in Regions introduced starting on December 23, 2023

Before you set up interface VPC endpoints for Amazon ECS, be aware of the following considerations:

- You must have the following Region-specific VPC endpoints:

  > **Note**
  >
  > If you do not configure all of the endpoints, your traffic will go over the public endpoints, not your VPC endpoint.

  - `com.amazonaws.region.ecs-agent`
  - `com.amazonaws.region.ecs-telemetry`
  - `com.amazonaws.region.ecs`

  For example, the Canada West (Calgary) (ca-west-1) Region needs the following VPC endpoints:

  - `com.amazonaws.ca-west-1.ecs-agent`
  - `com.amazonaws.ca-west-1.ecs-telemetry`
  - `com.amazonaws.ca-west-1.ecs`

- When you use a template to create AWS resources in the new Region and the template was copied from a Region introduced before December 23, 2023, depending on the copy-from Region, perform one of the following operations.

  For example, the copy-from Region is US East (N. Virginia) (us-east-1). The copy-to Region is Canada West (Calgary) (ca-west-1).

Configuration	Action	
The copied-from Region does not have any VPC endpoints.	Create all three VPC endpoints for the new Region (for example, `com.amazonaws.ca-west-1.ecs-agent`).	
The copied-from Region contains Region-specific VPC endpoints.	a. Create all three VPC endpoints for the new Region (for example, `com.amazonaws.ca-west-1.ecs-agent`). b. Delete all three VPC endpoints for the copy-from Region (for example, `com.amazonaws.us-east-1.ecs-agent`).	

Considerations for Amazon ECS VPC endpoints for the Fargate launch type

When there is an VPC endpoint for `ecr.dkr` and `ecr.api` in the same VPC where a Fargate task is deployed into, it will use the VPC endpoint. If there is no VPC endpoint, it will use the Fargate interface.

Before you set up interface VPC endpoints for Amazon ECS, be aware of the following considerations:

- Tasks using the Fargate launch type don't require the interface VPC endpoints for Amazon ECS, but you might need interface VPC endpoints for Amazon ECR, Secrets Manager, or Amazon CloudWatch Logs described in the following points.
  - To allow your tasks to pull private images from Amazon ECR, you must create the interface VPC endpoints for Amazon ECR. For more information, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.

    > **Important**
    >
    > If you configure Amazon ECR to use an interface VPC endpoint, you can create a task execution role that includes condition keys to restrict access to a specific VPC or VPC endpoint. For more information, see Fargate tasks pulling Amazon ECR images over interface endpoints permissions.

  - To allow your tasks to pull sensitive data from Secrets Manager, you must create the interface VPC endpoints for Secrets Manager. For more information, see Using Secrets Manager with VPC Endpoints in the *AWS Secrets Manager User Guide*.
  - If your VPC doesn't have an internet gateway and your tasks use the `awslogs` log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see Using CloudWatch Logs with Interface VPC Endpoints in the *Amazon CloudWatch Logs User Guide*.
- VPC endpoints currently don't support cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to Amazon ECS. For example, assume that you want to run tasks in US East (N. Virginia). Then, you must create the Amazon ECS VPC endpoint in US East (N. Virginia). An Amazon ECS VPC endpoint created in any other region can't run tasks in US East (N. Virginia).

- VPC endpoints only support Amazon-provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see DHCP Options Sets in the *Amazon VPC User Guide*.
- The security group attached to the VPC endpoint must allow incoming connections on TCP port 443 from the private subnet of the VPC.
- Service Connect management of the Envoy proxy uses the `com.amazonaws.region.ecs-agent` VPC endpoint. When you don't use the VPC endpoints, Service Connect management of the Envoy proxy uses the `ecs-sc` endpoint in that Region. For a list of the Amazon ECS endpoints in each Region, see Amazon ECS endpoints and quotas.

Considerations for Amazon ECS VPC endpoints for the EC2 launch type

Before you set up interface VPC endpoints for Amazon ECS, be aware of the following considerations:

- Tasks using the EC2 launch type require that the container instances that they're launched on to run version `1.25.1` or later of the Amazon ECS container agent. For more information, see Amazon ECS Linux container instance management.
- To allow your tasks to pull sensitive data from Secrets Manager, you must create the interface VPC endpoints for Secrets Manager. For more information, see Using Secrets Manager with VPC Endpoints in the *AWS Secrets Manager User Guide*.
- If your VPC doesn't have an internet gateway and your tasks use the `awslogs` log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see Using CloudWatch Logs with Interface VPC Endpoints in the *Amazon CloudWatch Logs User Guide*.
- VPC endpoints currently don't support cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to Amazon ECS. For example, assume that you want to run tasks in US East (N. Virginia). Then, you must create the Amazon ECS VPC endpoint in US East (N. Virginia). An Amazon ECS VPC endpoint created in any other Region can't run tasks in US East (N. Virginia).
- VPC endpoints only support Amazon-provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see DHCP Options Sets in the *Amazon VPC User Guide*.
- The security group attached to the VPC endpoint must allow incoming connections on TCP port 443 from the private subnet of the VPC.

Understanding Amazon ECS endpoint naming patterns

It's important to understand that the Amazon ECS agent may make requests to endpoints with numbered suffixes, such as:

- `ecs-a-1.region.amazonaws.com`, `ecs-a-2.region.amazonaws.com`, etc. for agent endpoints
- `ecs-t-1.region.amazonaws.com`, `ecs-t-2.region.amazonaws.com`, etc. for telemetry endpoints

This behavior occurs because the Amazon ECS agent uses the DiscoverPollEndpoint API to dynamically determine which specific endpoint to connect to. If your VPC endpoints don't properly handle these numbered endpoint variations, the agent will fall back to using public endpoints, even if you've configured VPC endpoints for the base names.

The role of DiscoverPollEndpoint API

The DiscoverPollEndpoint API is used by the Amazon ECS agent to discover the appropriate endpoint to poll for tasks. When the agent calls this API, it receives a specific endpoint URL that may include a numbered suffix. To ensure your VPC endpoints work correctly, your network configuration must allow the agent to:

1. Access the DiscoverPollEndpoint API
2. Connect to the returned endpoint URLs, including those with numbered suffixes

If you're troubleshooting VPC endpoint connectivity issues, verify that your agent can reach both the base endpoints and any numbered variations that might be returned by the DiscoverPollEndpoint API.

Creating the VPC Endpoints for Amazon ECS

To create the VPC endpoint for the Amazon ECS service, use the Access an AWS service using an interface VPC endpoint procedure in the *Amazon VPC User Guide* to create the following endpoints. If you have existing container instances within your VPC, you should create the endpoints in the order that they're listed. If you plan on creating your container instances after your VPC endpoint is created, the order doesn't matter.

> **Note**
>
> If you do not configure all of the endpoints, your traffic will go over the public endpoints, not your VPC endpoint.
> When you create endpoints, Amazon ECS also creates a private DNS name for the endpoint. For example, `ecs-a.region.amazonaws.com` for ecs-agent and `ecs-t.region.amazonaws.com` for ecs-telemetry.

- `com.amazonaws.`*`region`*`.ecs-agent`
- `com.amazonaws.`*`region`*`.ecs-telemetry`
- `com.amazonaws.`*`region`*`.ecs`

> **Note**
>
> *`region`* represents the Region identifier for an AWS Region supported by Amazon ECS, such as `us-east-2` for the US East (Ohio) Region.

The `ecs-agent` endpoint uses the `ecs:poll` API, and the `ecs-telemetry` endpoint uses the `ecs:poll` and `ecs:StartTelemetrySession` API.

If you have existing tasks that are using the EC2 launch type, after you have created the VPC endpoints, each container instance needs to pick up the new configuration. For this to happen, you must either reboot each container instance or restart the Amazon ECS container agent on each container instance. To restart the container agent, do the following.

To restart the Amazon ECS container agent

1. Log in to your container instance via SSH.
2. Stop the container agent.

   ```
   sudo docker stop ecs-agent
   ```

3. Start the container agent.

   ```
   sudo docker start ecs-agent
   ```

After you have created the VPC endpoints and restarted the Amazon ECS container agent on each container instance, all newly launched tasks pick up the new configuration.

Creating a VPC endpoint policy for Amazon ECS

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon ECS. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Amazon ECS actions

The following is an example of an endpoint policy for Amazon ECS. When attached to an endpoint, this policy grants access to permission to create and list clusters. The `CreateCluster` and `ListClusters` actions do not accept any resources, so the resource definition is set to * for all resources.

```
{
   "Statement":[
    {
      "Principal":"*",
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:ListClusters"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

AWS shared responsibility model for Amazon ECS

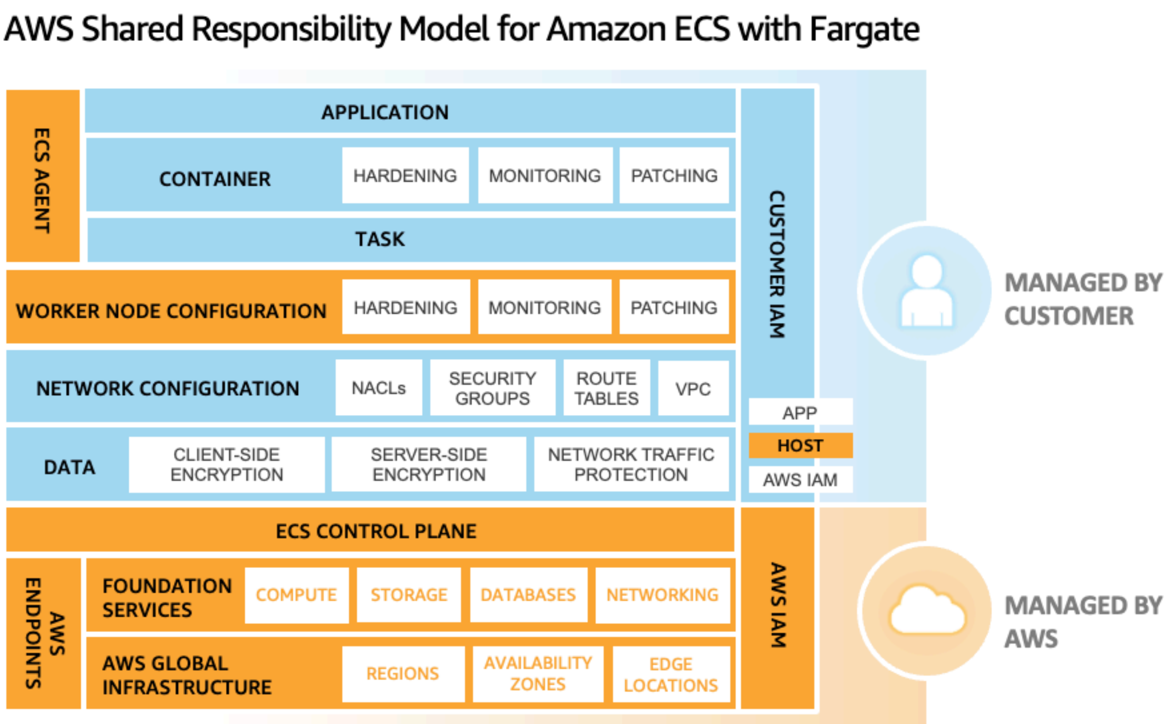
Security and Compliance is a shared responsibility between AWS and the customer. This shared model can help relieve the customer's operational burden as AWS operates, manages and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the service operates. The customer assumes responsibility and management of the guest operating system (including updates and security patches), other associated application software as well as the configuration of the AWS provided security group firewall. Customers should carefully consider the services they choose as their responsibilities vary depending on the services used, the integration of those services into their IT environment, and applicable laws and regulations. The nature of this shared responsibility also provides the flexibility and customer control that permits the deployment.

Fargate launch type

The following illustration shows the shared responsibility model for the Fargate launch type. Fargate runs each workload in an isolated hardware virtualized environment. As a result, each task gets dedicated infrastructure capacity. Containerized workloads running on Fargate do not share an operating system, Linux kernel, network interface, ephemeral storage, CPU, or memory with other tasks. When using Fargate, customers are not responsible for securing the compute infrastructure that runs their containers. Fargate will provision and patch the infrastructure upon which customer workloads run. For more information, see Task retirement and maintenance for AWS Fargate on Amazon ECS .

You are responsible for managing the following resources:

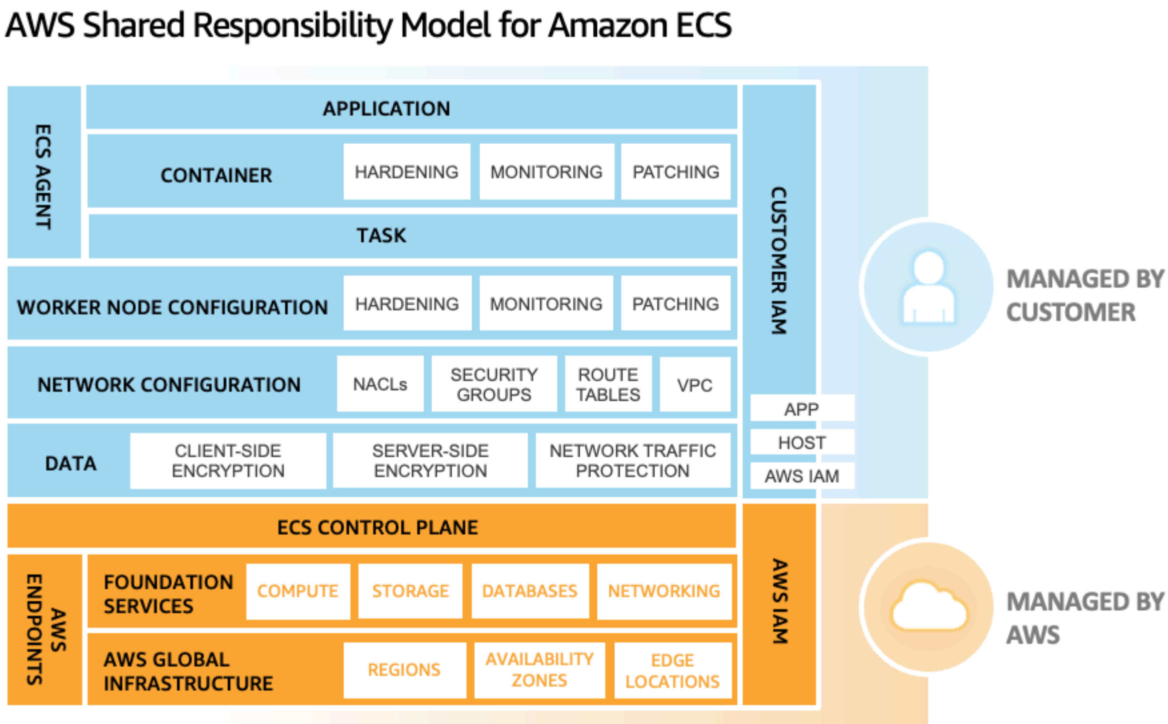
- Network configuration including VPC, NACLs, security groups, and route tables
- Client and service storage encryption. For more information, see Storage options for Amazon ECS tasks.
- Container images. For more information, see Amazon ECS task and container security best practices.
- IAM permissions for the applications by using the task role. For more information, see Amazon ECS task IAM role.



EC2 launch type

The following illustration shows the shared responsibility for the EC2 launch type. When you run tasks on EC2 instances you are responsible for maintaining your EC2 instances in addition to the following resources:

- The Amazon ECS agent.
- • The EC2 instance AMI, including patching and hardening.
- Network configuration including VPC, NACLs, security groups, and route tables.
- Client and service storage encryption. For more information, see Storage options for Amazon ECS tasks.
- Container images. For more information, see Amazon ECS task and container security best practices.
- IAM permissions for the applications by using the task role. For more information, see Amazon ECS task IAM role.



Network security best practices for Amazon ECS

Network security is a broad topic that encompasses several subtopics. These include encryption-in-transit, network segmentation and isolation, firewalling, traffic routing, and observability.

Encryption in transit

Encrypting network traffic prevents unauthorized users from intercepting and reading data when that data is transmitted across a network. With Amazon ECS, network encryption can be implemented in any of the following ways.

- **Using Nitro instances:**

  By default, traffic is automatically encrypted between the following Nitro instance types: C5n, G4, I3en, M5dn, M5n, P3dn, R5dn, and R5n. Traffic isn't encrypted when it's routed through a transit gateway, load balancer, or similar intermediary.

  - Encryption in transit
  - What's new announcement from 2019
  - This talk from re:Inforce 2019
- **Using Server Name Indication (SNI) with an Application Load Balancer:**

The Application Load Balancer (ALB) and Network Load Balancer (NLB) support Server Name Indication (SNI). By using SNI, you can put multiple secure applications behind a single listener. For this, each has its own TLS certificate. We recommend that you provision certificates for the load balancer using AWS Certificate Manager (ACM) and then add them to the listener's certificate list. The AWS load balancer uses a smart certificate selection algorithm with SNI. If the hostname that's provided by a client matches a single certificate in the certificate list, the load balancer chooses that certificate. If a hostname that's provided by a client matches multiple certificates in the list, the load balancer selects a certificate that the client can support. Examples include self-signed certificate or a certificate generated through the ACM.

  - SNI with Application Load Balancer
  - SNI with Network Load Balancer

- **End-to-end encryption with TLS certificates:**

  This involves deploying a TLS certificate with the task. This can either be a self-signed certificate or a certificate from a trusted certificate authority. You can obtain the certificate by referencing a secret for the certificate. Otherwise, you can choose to run a container that issues a Certificate Signing Request (CSR) to ACM and then mounts the resulting secret to a shared volume.

  - Maintaining transport layer security all the way to your containers using the Network Load Balancer with Amazon ECS part 1
  - Maintaining Transport Layer Security (TLS) all the way to your container part 2: Using AWS Private Certificate Authority

Task networking

The following recommendations are in consideration of how Amazon ECS works. Amazon ECS doesn't use an overlay network. Instead, tasks are configured to operate in different network modes. For example, tasks that are configured to use `bridge` mode acquire a non-routable IP address from a Docker network that runs on each host. Tasks that are configured to use the `awsvpc` network mode acquire an IP address from the subnet of the host. Tasks that are configured with `host` networking use the host's network interface. `awsvpc` is the preferred network mode. This is because it's the only mode that you can use to assign security groups to tasks. It's also the only mode that's available for AWS Fargate tasks on Amazon ECS.

Security groups for tasks

We recommend that you configure your tasks to use the `awsvpc` network mode. After you configure your task to use this mode, the Amazon ECS agent automatically provisions and attaches an Elastic Network Interface (ENI) to the task. When the ENI is provisioned, the task is enrolled in an AWS security group. The security group acts as a virtual firewall that you can use to control inbound and outbound traffic.

If you use a custom firewall with tasks or services, add an outbound rule to allow traffic for the Amazon ECS agent management endpoints ("`ecs-a-*.`*`region`*`.amazonaws.com`"), telemetry endpoints ("`ecs-t-*.`*`region`*`.amazonaws.com`"), and the Service Connect Envoy management endpoints ("`ecs-sc.`*`region`*`.api.aws`").

AWS PrivateLink and Amazon ECS

AWS PrivateLink is a networking technology that allows you to create private endpoints for different AWS services, including Amazon ECS. The endpoints are required in sandboxed environments where there is no Internet Gateway (IGW) attached to the Amazon VPC and no alternative routes to the Internet. Using AWS PrivateLink ensures that calls to the Amazon ECS service stay within the Amazon VPC and do not traverse the internet. For instructions on how to create AWS PrivateLink endpoints for Amazon ECS and other related services, see Amazon ECS interface Amazon VPC endpoints.

Important

AWS Fargate tasks don't require an AWS PrivateLink endpoint for Amazon ECS.

Amazon ECR and Amazon ECS both support endpoint policies. These policies allow you to refine access to a service's APIs. For example, you could create an endpoint policy for Amazon ECR that only allows images to be pushed to registries in particular AWS accounts. A policy like this could be used to prevent data from being exfiltrated through container images while still allowing users to push to authorized Amazon ECR registries. For more information, see Use VPC endpoint policies.

The following policy allows all AWS principals in your account to perform all actions against only your Amazon ECR repositories:

```
{
```

```
  "Statement": [
    {
      "Sid": "LimitECRAccess",
      "Principal": "*",
      "Action": "*",
      "Effect": "Allow",
      "Resource": "arn:aws:ecr:region:account_id:repository/*"
    },
  ]
}
```

You can enhance this further by setting a condition that uses the new `PrincipalOrgID` property. This prevents pushing and pulling of images by an IAM principal that isn't part of your AWS Organizations. For more information, see aws:PrincipalOrgID.

We recommended applying the same policy to both the `com.amazonaws.region.ecr.dkr` and the `com.amazonaws.region.ecr.api` endpoints.

Container agent settings

The Amazon ECS container agent configuration file includes several environment variables that relate to network security. `ECS_AWSVPC_BLOCK_IMDS` and `ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST` are used to block a task's access to Amazon EC2 metadata. `HTTP_PROXY` is used to configure the agent to route through a HTTP proxy to connect to the internet. For instructions on configuring the agent and the Docker runtime to route through a proxy, see HTTP Proxy Configuration.

> **Important**
>
> These settings aren't available when you use AWS Fargate.

Network security recommendations

We recommend that you do the following when setting up your Amazon VPC, load balancers, and network.

Use network encryption where applicable with Amazon ECS

You should use network encryption where applicable. Certain compliance programs, such as PCI DSS, require that you encrypt data in transit if the data contains cardholder data. If your workload has similar requirements, configure network encryption.

Modern browsers warn users when connecting to insecure sites. If your service is fronted by a public facing load balancer, use TLS/SSL to encrypt the traffic from the client's browser to the load balancer and re-encrypt to the backend if warranted.

Use `awsvpc` network mode and security groups to control traffic between tasks and other resources in Amazon ECS

You should use `awsvpc` network mode and security groups when you need to control traffic between tasks and between tasks and other network resources. If your service is behind an ALB, use security groups to only allow inbound traffic from other network resources using the same security group as your ALB. If your application is behind an NLB, configure the task's security group to only allow inbound traffic from the Amazon VPC CIDR range and the static IP addresses assigned to the NLB.

Security groups should also be used to control traffic between tasks and other resources within the Amazon VPC such as Amazon RDS databases.

Create Amazon ECS clusters in separate Amazon VPCs when network traffic needs to be strictly isolated

You should create clusters in separate Amazon VPCs when network traffic needs to be strictly isolated. Avoid running workloads that have strict security requirements on clusters with workloads that don't have to adhere to those requirements. When strict network isolation is mandatory, create clusters in separate Amazon VPCs and selectively expose services to other Amazon VPCs using Amazon VPC endpoints. For more information, see VPC endpoints.

Configure AWS PrivateLink endpoints when warranted for Amazon ECS

You should configure AWS PrivateLink endpoints when warranted. If your security policy prevents you from attaching an Internet Gateway (IGW) to your Amazon VPCs, configure AWS PrivateLink endpoints for Amazon ECS and other services such as Amazon ECR, AWS Secrets Manager, and Amazon CloudWatch.

Use Amazon VPC Flow Logs to analyze the traffic to and from long-running tasks in Amazon ECS

You should use Amazon VPC Flow Logs to analyze the traffic to and from long-running tasks. Tasks that use `awsvpc` network mode get their own ENI. Doing this, you can monitor traffic that goes to and from individual tasks using Amazon VPC Flow Logs. A recent update to Amazon VPC Flow Logs (v3), enriches the logs with traffic metadata including the vpc ID, subnet ID, and the instance ID. This metadata can be used to help narrow an investigation. For more information, see [Amazon VPC Flow Logs](#).

> **Note**
>
> Because of the temporary nature of containers, flow logs might not always be an effective way to analyze traffic patterns between different containers or containers and other network resources.

Amazon ECS task and container security best practices

You should consider the container image as your first line of defense against an attack. An insecure, poorly constructed image can allow an attacker to escape the bounds of the container and gain access to the host. You should do the following to mitigate the risk of this happening.

We recommend that you do the following when setting up your tasks and containers.

Create minimal or use distroless images

Start by removing all extraneous binaries from the container image. If you're using an unfamiliar image from Amazon ECR Public Gallery, inspect the image to refer to the contents of each of the container's layers. You can use an application such as Dive to do this.

Alternatively, you can use **distroless** images that only include your application and its runtime dependencies. They don't contain package managers or shells. Distroless images improve the "signal to noise of scanners and reduces the burden of establishing provenance to just what you need." For more information, see the GitHub documentation on distroless.

Docker has a mechanism for creating images from a reserved, minimal image known as **scratch**. For more information, see Creating a simple parent image using **scratch** in the Docker documentation.

With languages like Go, you can create a static linked binary and reference it in your Dockerfile. The following example shows how you can accomplish this.

```
############################
# STEP 1 build executable binary
############################
FROM golang:alpine AS builder
# Install git.
# Git is required for fetching the dependencies.
RUN apk update && apk add --no-cache git
WORKDIR $GOPATH/src/mypackage/myapp/
COPY . .
# Fetch dependencies.
# Using go get.
RUN go get -d -v
# Build the binary.
RUN go build -o /go/bin/hello
############################
# STEP 2 build a small image
############################
FROM scratch
# Copy our static executable.
COPY --from=builder /go/bin/hello /go/bin/hello
# Run the hello binary.
ENTRYPOINT ["/go/bin/hello"]
This creates a container image that consists of your application and nothing else,
 making it extremely secure.
```

The previous example is also an example of a multi-stage build. These types of builds are attractive from a security standpoint because you can use them to minimize the size of the final image pushed to your container registry. Container images devoid of build tools and other extraneous binaries improves your security posture by reducing the attack surface of the image. For more information about multi-stage builds, see Multi-stage builds in the Docker documentation.

Scan your images for vulnerabilities

Similar to their virtual machine counterparts, container images can contain binaries and application libraries with vulnerabilities or develop vulnerabilities over time. The best way to safeguard against exploits is by regularly scanning your images with an image scanner.

Amazon ECR provides two versions of basic scanning that use the Common Vulnerabilities and Exposures (CVEs) database:

- **AWS native basic scanning –** Uses AWS native technology, which is now GA and recommended. This improved basic scanning is designed to provide customers with better scanning results and vulnerability detection across a broad set of popular operating systems. This allows customers to further strengthen the security of their container images. All new customer registries are opted into this improved version by default.
- **Clair basic scanning –** The previous basic scanning version which uses the open-source Clair project and is deprecated. For more information about Clair, see Clair on GitHub.

Both AWS native and Clair basic scanning are supported in all regions listed in AWS Services by Region, except for those that were added after September, 2024. Because Clair support is deprecated, Clair will not be supported in new regions as they are added and will no longer be supported in all regions as of October 1, 2025.

Amazon ECR uses the severity for a CVE from the upstream distribution source if available. Otherwise, the Common Vulnerability Scoring System (CVSS) score is used. The CVSS score can be used to obtain the NVD vulnerability severity rating. For more information, see NVD Vulnerability Severity Ratings.

You can also see the results of a scan from within the Amazon ECR console or by calling the DescribeImageScanFindings API. Images with a `HIGH` or `CRITICAL` vulnerability should be deleted or rebuilt. If an image that has been deployed develops a vulnerability, it should be replaced as soon as possible.

Docker Desktop Edge version 2.3.6.0 or later can scan local images. The scans are powered by Snyk, an application security service. When vulnerabilities are discovered, Snyk identifies the layers and dependencies with the vulnerability in the Dockerfile. It also recommends safe alternatives like using a slimmer base image with fewer vulnerabilities or upgrading a particular package to a newer version. By using Docker scan, developers can resolve potential security issues before pushing their images to the registry.

- Automating image compliance using Amazon ECR and AWS Security Hub explains how to surface vulnerability information from Amazon ECR in AWS Security Hub and automate remediation by blocking access to vulnerable images.

Remove special permissions from your images

The access rights flags `setuid` and `setgid` allow running an executable with the permissions of the owner or group of the executable. Remove all binaries with these access rights from your image as these binaries can be used to escalate privileges. Consider removing all shells and utilities like `nc` and `curl` that can be used for malicious purposes. You can find the files with `setuid` and `setgid` access rights by using the following command.

```
find / -perm /6000 -type f -exec ls -ld {} \;
```

To remove these special permissions from these files, add the following directive to your container image.

```
RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true
```

Create a set of curated images

Rather than allowing developers to create their own images, create a set of vetted images for the different application stacks in your organization. By doing so, developers can forego learning how to compose Dockerfiles and concentrate on writing code. As changes are merged into your codebase, a CI/CD pipeline can automatically compile the asset and then store it in an artifact repository. And, last, copy the artifact into the appropriate image before pushing it to a Docker registry such as Amazon ECR. At the very least you should create a set of base images that developers can create their own Dockerfiles from. You should avoid pulling images from Docker Hub. You don't always know what is in the image and about a fifth of the top 1000 images have vulnerabilities. A list of those images and their vulnerabilities can be found at https://vulnerablecontainers.org/.

Scan application packages and libraries for vulnerabilities

Use of open source libraries is now common. As with operating systems and OS packages, these libraries can have vulnerabilities. As part of the development lifecycle these libraries should be scanned and updated when critical vulnerabilities are found.

Docker Desktop performs local scans using Snyk. It can also be used to find vulnerabilities and potential licensing issues in open source libraries. It can be integrated directly into developer workflows giving you the ability to mitigate risks posed by open source libraries. For more information, see the following topics:

- Open Source Application Security Tools includes a list of tools for detecting vulnerabilities in applications.

Perform static code analysis

You should perform static code analysis before building a container image. It's performed against your source code and is used to identify coding errors and code that could be exploited by a malicious actor, such as fault injections. You can use Amazon Inspector. For more information, see Scanning Amazon ECR container images with Amazon Inspector in the *Amazon Inspector User Guide*.

Run containers as a non-root user

You should run containers as a non-root user. By default, containers run as the `root` user unless the `USER` directive is included in your Dockerfile. The default Linux capabilities that are assigned by Docker restrict the actions that can be run as `root`, but only marginally. For example, a container running as `root` is still not allowed to access devices.

As part of your CI/CD pipeline you should lint Dockerfiles to look for the `USER` directive and fail the build if it's missing. For more information, see the following topics:

- Dockerfile-lint is an open-source tool from RedHat that can be used to check if the file conforms to best practices.
- Hadolint is another tool for building Docker images that conform to best practices.

Use a read-only root file system

You should use a read-only root file system. A container's root file system is writable by default. When you configure a container with a `RO` (read-only) root file system it forces you to explicitly define where data can be persisted. This reduces your attack surface because the container's file system can't be written to unless permissions are specifically granted.

> **Note**
>
> Having a read-only root file system can cause issues with certain OS packages that expect to be able to write to the filesystem. If you're planning to use read-only root file systems, thoroughly test beforehand.

Configure tasks with CPU and Memory limits (Amazon EC2)

You should configure tasks with CPU and memory limits to minimize the following risk. A task's resource limits set an upper bound for the amount of CPU and memory that can be reserved by all the containers within a task. If no limits are set, tasks have access to the host's CPU and memory. This can cause issues where tasks deployed on a shared host can starve other tasks of system resources.

> **Note**
>
> Amazon ECS on AWS Fargate tasks require you to specify CPU and memory limits because it uses these values for billing purposes. One task hogging all of the system resources isn't an issue for Amazon ECS Fargate because each task is run on its own dedicated instance. If you don't specify a memory limit, Amazon ECS allocates a minimum of 4MB to each container. Similarly, if no CPU limit is set for the task, the Amazon ECS container agent assigns it a minimum of 2 CPUs.

Use immutable tags with Amazon ECR

With Amazon ECR, you can and should use configure images with immutable tags. This prevents pushing an altered or updated version of an image to your image repository with an identical tag. This protects against an attacker pushing a compromised version of an image over your image with the same tag. By using immutable tags, you effectively force yourself to push a new image with a different tag for each change.

Avoid running containers as privileged (Amazon EC2)

You should avoid running containers as privileged. For background, containers run as `privileged` are run with extended privileges on the host. This means the container inherits all of the Linux capabilities assigned to `root` on the host. It's use should be severely restricted or forbidden. We advise setting the Amazon ECS container agent environment variable `ECS_DISABLE_PRIVILEGED` to `true` to prevent containers from running as `privileged` on particular hosts if `privileged` isn't needed. Alternatively you can use AWS Lambda to scan your task definitions for the use of the `privileged` parameter.

> **Note**
>
> Running a container as `privileged` isn't supported on Amazon ECS on AWS Fargate.

Remove unnecessary Linux capabilities from the container

The following is a list of the default Linux capabilities assigned to Docker containers. For more information about each capability, see [Overview of Linux Capabilities](#).

```
CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID, CAP_KILL,
CAP_SETGID, CAP_SETUID, CAP_SETPCAP, CAP_NET_BIND_SERVICE,
CAP_NET_RAW, CAP_SYS_CHROOT, CAP_MKNOD, CAP_AUDIT_WRITE,
CAP_SETFCAP
```

If a container doesn't require all of the Docker kernel capabilities listed above, consider dropping them from the container. For more information about each Docker kernel capability, see [KernelCapabilities](#). You can find out which capabilities are in use by doing the following:

- Install the OS package [libcap-ng](#) and run the `pscap` utility to list the capabilities that each process is using.
- You can also use [capsh](#) to decipher which capabilities a process is using.

Use a customer managed key (CMK) to encrypt images pushed to Amazon ECR

You should use a customer managed key (CMK) to encrypt images that are pushed to Amazon ECR. Images that are pushed to Amazon ECR are automatically encrypted at rest with a AWS Key Management Service (AWS KMS) managed key. If you would rather use your own key, Amazon ECR now supports AWS KMS encryption with customer managed keys (CMK). Before enabling server side encryption with a CMK, review the Considerations listed in the documentation on encryption at rest.

Tutorials for Amazon ECS

The following tutorials show you how to perform common tasks when using Amazon ECS.

You can use any of the following tutorials to learn more about getting started with Amazon ECS.

Tutorial overview	Learn more
Get started with Amazon ECS on Fargate.	Learn how to create an Amazon ECS Linux task for the Fargate launch type
Get started with Windows containers on Fargate.	Learn how to create an Amazon ECS Windows task for the Fargate launch type
Get started with Windows containers for the EC2 launch type.	Learn how to create an Amazon ECS Windows task for the EC2 launch type

You can use any of the following tutorials to deploy tasks on Amazon ECS using the AWS CLI

Tutorial overview	Learn more
Create a Linux task for the Fargate launch type.	Creating an Amazon ECS Linux task for the Fargate launch type with the AWS CLI
Create a Windows task for the Fargate launch type.	Creating an Amazon ECS Windows task for the Fargate launch type with the AWS CLI
Create a Linux task for the EC2 launch type.	Creating an Amazon ECS task for the EC2 launch type with the AWS CLI

You can use any of the following tutorials to learn more about monitoring and logging.

Tutorial overview	Learn more
Set up a simple Lambda function that listens for task events and writes them out to a CloudWatch Logs log stream.	Configuring Amazon ECS to listen for CloudWatch Events events
Configure an Amazon EventBridge event rule that only captures task events where the task has stopped running because one of its essential containers has terminated.	Sending Amazon Simple Notification Service alerts for Amazon ECS task stopped events
Concatenate log messages that originally belong to one context but were split across multiple records or log lines.	Concatenating multiline or stack-trace Amazon ECS log messages
Deploy Fluent Bit containers on their Windows instances running in Amazon ECS to stream logs generated by the Windows tasks to Amazon CloudWatch for centralized logging.	Deploying Fluent Bit on Amazon ECS Windows containers

You can use any of the following tutorials to learn more about how to use Active Directory authentication with group Managed Service Account on Amazon ECS.

Tutorial overview	Learn more
Use group Managed Service Account with Linux containers on EC2.	Using gMSA for EC2 Linux containers on Amazon ECS
Use group Managed Service Account with Windows containers on EC2.	Learn how to use gMSAs for EC2 Windows containers for Amazon ECS
Use group Managed Service Account with Linux containers on Fargate.	Using gMSA for Linux containers on Fargate
Create a task that runs a Windows container that has credentials to access Active Directory with domainles s group Managed Service Account.	Using Amazon ECS Windows containers with domainless gMSA using the AWS CLI

Creating an Amazon ECS Linux task for the Fargate launch type with the AWS CLI

The following steps help you set up a cluster, register a task definition, run a Linux task, and perform other common scenarios in Amazon ECS with the AWS CLI. Use the latest version of the AWS CLI. For more information on how to upgrade to the latest version, see Installing or updating to the latest version of the AWS CLI.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

Topics

Prerequisites

This tutorial assumes that the following prerequisites have been completed.

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, Installing or updating to the latest version of the AWS CLI.
- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Amazon ECR Public so your task must have internet access. To give your task a route to the internet, use one of the following options.
  - Use a private subnet with a NAT gateway that has an elastic IP address.
  - Use a public subnet and assign a public IP address to the task.

  For more information, see the section called “Create a virtual private cloud”.

  For information about security groups and rules, see, Default security groups for your VPCs and Example rules in the *Amazon Virtual Private Cloud User Guide*.
- If you follow this tutorial using a private subnet, you can use Amazon ECS Exec to directly interact with your container and test the deployment. You will need to create a task IAM role to use ECS Exec. For more information on the task IAM role and other prerequisites, see Monitor Amazon ECS containers with Amazon ECS Exec.

- (Optional) AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell?](#) in the *AWS CloudShell User Guide*.

Step 1: Create a Cluster

By default, your account receives a `default` cluster.

> **Note**
>
> The benefit of using the `default` cluster that is provided for you is that you don't have to specify the `--cluster cluster_name` option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster cluster_name` for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{
    "cluster": {
        "status": "ACTIVE",
        "defaultCapacityProviderStrategy": [],
        "statistics": [],
        "capacityProviders": [],
        "tags": [],
        "clusterName": "fargate-cluster",
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
```

```
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

Step 2: Register a Linux Task Definition

Before you can run a task on your ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that creates a PHP web app using the httpd container image hosted on Docker Hub. For more information about the available task definition parameters, see Amazon ECS task definitions. For this tutorial, the `taskRoleArn` is only needed if you are deploying the task in a private subnet and wish to test the deployment. Replace the `taskRoleArn` with the IAM task role you created to use ECS Exec as mentioned in Prerequisites.

```
{
        "family": "sample-fargate",
        "networkMode": "awsvpc",
        "taskRoleArn": "arn:aws:iam::aws_account_id:role/execCommandRole",
        "containerDefinitions": [
            {
                "name": "fargate-app",
                "image": "public.ecr.aws/docker/library/httpd:latest",
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "essential": true,
                "entryPoint": [
                    "sh",
                    "-c"
                ],
                "command": [
                    "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
                ]
```

```
            }
        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
        "memory": "512"
}
```

Save the task definition JSON as a file and pass it with the `--cli-input-json file://path_to_file.json` option.

To use a JSON file for container definitions:

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

The **register-task-definition** command returns a description of the task definition after it completes its registration.

Step 3: List Task Definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the `family` and `revision` values that you can use together when calling **run-task** or **start-task**.

```
aws ecs list-task-definitions
```

Output:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:1"
    ]
}
```

Step 4: Create a Service

After you have registered a task for your account, you can create a service for the registered task in your cluster. For this example, you create a service with one instance of the `sample-fargate:1` task definition running in your cluster. The task requires a route to the internet, so there are two ways you can achieve this. One way is to use a private subnet configured with a NAT gateway with an elastic IP address in a public subnet. Another way is to use a public subnet and assign a public IP address to your task. We provide both examples below.

Example using a private subnet. The `enable-execute-command` option is needed to use Amazon ECS Exec.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234]}" --enable-execute-command
```

Example using a public subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234],assignPublicIp=ENABLED}"
```

The **create-service** command returns a description of the task definition after it completes its registration.

Step 5: List Services

List the services for your cluster. You should see the service that you created in the previous section. You can take the service name or the full ARN that is returned from this command and use it to describe the service later.

```
aws ecs list-services --cluster fargate-cluster
```

Output:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-cluster/fargate-service"
    ]
}
```

Step 6: Describe the Running Service

Describe the service using the service name retrieved earlier to get more information about the task.

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

If successful, this will return a description of the service failures and services. For example, in the `services` section, you will find information on deployments, such as the status of the tasks as running or pending. You may also find information on the task definition, the network configuration and time-stamped events. In the failures section, you will find information on failures, if any, associated with the call. For troubleshooting, see Service Event Messages. For more information about the service description, see Describe Services.

```
{
    "services": [
        {
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "ENABLED"
                }
            },
            "launchType": "FARGATE",
            "enableECSManagedTags": false,
            "loadBalancers": [],
            "deploymentController": {
                "type": "ECS"
            },
            "desiredCount": 1,
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "createdAt": 1692283199.771,
```

```
            "schedulingStrategy": "REPLICA",
            "placementConstraints": [],
            "deployments": [
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
                            ],
                            "assignPublicIp": "ENABLED"
                        }
                    },
                    "pendingCount": 0,
                    "launchType": "FARGATE",
                    "createdAt": 1692283199.771,
                    "desiredCount": 1,
                    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-
definition/sample-fargate:1",
                    "updatedAt": 1692283199.771,
                    "platformVersion": "1.4.0",
                    "id": "ecs-svc/9223370526043414679",
                    "runningCount": 0
                }
            ],
            "serviceName": "fargate-service",
            "events": [
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                    "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                    "createdAt": 1510811841.408
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                    "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                    "createdAt": 1510811601.938
                },
                {
```

```
                    "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                    "createdAt": 1510811364.691
                }
            ],
            "runningCount": 0,
            "status": "ACTIVE",
            "serviceRegistries": [],
            "pendingCount": 0,
            "createdBy": "arn:aws:iam::aws_account_id:user/user_name",
            "platformVersion": "LATEST",
            "placementStrategy": [],
            "propagateTags": "NONE",
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate:1"
        }
    ],
    "failures": []
}
```

Step 7: Test

Testing task deployed using public subnet

Describe the task in the service so that you can get the Elastic Network Interface (ENI) for the task.

First, get the task ARN.

```
aws ecs list-tasks --cluster fargate-cluster --service fargate-service
```

The output contains the task ARN.

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE
    ]
}
```

Describe the task and locate the ENI ID. Use the task ARN for the `tasks` parameter.

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-east-1:123456789012:task/service/EXAMPLE
```

The attachment information is listed in the output.

```
{
    "tasks": [
        {
            "attachments": [
                {
                    "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
                    "type": "ElasticNetworkInterface",
                    "status": "ATTACHED",
                    "details": [
                        {
                            "name": "subnetId",
                            "value": "subnetabcd1234"
                        },
                        {
                            "name": "networkInterfaceId",
                            "value": "eni-0fa40520aeEXAMPLE"
                        },
                    ]
                }
…
}
```

Describe the ENI to get the public IP address.

```
aws ec2 describe-network-interfaces --network-interface-id  eni-0fa40520aeEXAMPLE
```

The public IP address is in the output.

```
{
    "NetworkInterfaces": [
        {
            "Association": {
                "IpOwnerId": "amazon",
                "PublicDnsName": "ec2-34-229-42-222.compute-1.amazonaws.com",
                "PublicIp": "198.51.100.2"
            },
```

```
…
}
```

Enter the public IP address in your web browser and you should see a webpage that displays the **Amazon ECS** sample application.

Testing task deployed using private subnet

Describe the task and locate `managedAgents` to verify that the `ExecuteCommandAgent` is running. Note the `privateIPv4Address` for later use.

```
aws ecs describe-tasks --cluster fargate-cluster --tasks arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE
```

The managed agent information is listed in the output.

```
{
     "tasks": [
        {
            "attachments": [
                {
                    "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
                    "type": "ElasticNetworkInterface",
                    "status": "ATTACHED",
                    "details": [
                        {
                            "name": "subnetId",
                            "value": "subnetabcd1234"
                        },
                        {
                            "name": "networkInterfaceId",
                            "value": "eni-0fa40520aeEXAMPLE"
                        },
                        {
                            "name": "privateIPv4Address",
                            "value": "10.0.143.156"
                        }
                    ]
                }
            ],
     ...
     "containers": [
```

```
        {
        ...
       "managedAgents": [
                       {
                           "lastStartedAt": "2023-08-01T16:10:13.002000+00:00",
                           "name": "ExecuteCommandAgent",
                           "lastStatus": "RUNNING"
                       }
               ],
       ...
   }
```

After verifying that the ExecuteCommandAgent is running, you can run the following command to run an interactive shell on the container in the task.

```
aws ecs execute-command --cluster fargate-cluster \
    --task  arn:aws:ecs:us-east-1:123456789012:task/fargate-service/EXAMPLE  \
    --container  fargate-app \
    --interactive \
    --command "/bin/sh"
```

After the interactive shell is running, run the following commands to install cURL.

```
apt update
```

```
apt install curl
```

After installing cURL, run the following command using the private IP address you obtained earlier.

```
curl 10.0.143.156
```

You should see the HTML equivalent of the **Amazon ECS** sample application webpage.

```
<html>
    <head>
     <title>Amazon ECS Sample App</title>
     <style>body {margin-top: 40px; background-color: #333;} </style>
    </head>
      <body>
      <div style=color:white;text-align:center>
```

```
        <h1>Amazon ECS Sample App</h1>
        <h2>Congratulations!</h2> <p>Your application is now running on a container in
  Amazon ECS.</p>
        </div>
        </body>
</html>
```

Step 8: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

Creating an Amazon ECS Windows task for the Fargate launch type with the AWS CLI

The following steps help you set up a cluster, register a task definition, run a Windows task, and perform other common scenarios in Amazon ECS with the AWS CLI. Ensure that you are using the latest version of the AWS CLI. For more information on how to upgrade to the latest version, see Installing or updating to the latest version of the AWS CLI.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

Topics

- Prerequisites

Prerequisites

This tutorial assumes that the following prerequisites have been completed.

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see Installing or updating to the latest version of the AWS CLI.
- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Docker Hub so your task must have internet access. To give your task a route to the internet, use one of the following options.
  - Use a private subnet with a NAT gateway that has an elastic IP address.
  - Use a public subnet and assign a public IP address to the task.

  For more information, see the section called “Create a virtual private cloud”.

  For information about security groups and rules, see, Default security groups for your VPCs and Example rules in the *Amazon Virtual Private Cloud User Guide*.
- (Optional) AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see What is AWS CloudShell? in the *AWS CloudShell User Guide*.

Step 1: Create a Cluster

By default, your account receives a `default` cluster.

> **Note**
>
> The benefit of using the `default` cluster that is provided for you is that you don't have to specify the `--cluster cluster_name` option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster cluster_name` for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{
    "cluster": {
        "status": "ACTIVE",
        "statistics": [],
        "clusterName": "fargate-cluster",
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

Step 2: Register a Windows Task Definition

Before you can run a Windows task on your Amazon ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that creates a web app. For more information about the available task definition parameters, see Amazon ECS task definitions.

```
{
    "containerDefinitions": [
        {
            "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file
 -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
```

```
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe
 w3svc"],
            "entryPoint": [
                "powershell",
                "-Command"
            ],
            "essential": true,
            "cpu": 2048,
            "memory": 4096,
            "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
            "name": "sample_windows_app",
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "memory": "4096",
    "cpu": "2048",
    "networkMode": "awsvpc",
    "family": "windows-simple-iis-2019-core",
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
    "runtimePlatform": {"operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"},
    "requiresCompatibilities": ["FARGATE"]
}
```

The above example JSON can be passed to the AWS CLI in two ways: You can save the task definition JSON as a file and pass it with the `--cli-input-json file://path_to_file.json` option.

To use a JSON file for container definitions:

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

The **register-task-definition** command returns a description of the task definition after it completes its registration.

Step 3: List task definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the `family` and `revision` values that you can use together when calling **run-task** or **start-task**.

```
aws ecs list-task-definitions
```

Output:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate-windows:1"
    ]
}
```

Step 4: Create a service

After you have registered a task for your account, you can create a service for the registered task in your cluster. For this example, you create a service with one instance of the `sample-fargate:1` task definition running in your cluster. The task requires a route to the internet, so there are two ways you can achieve this. One way is to use a private subnet configured with a NAT gateway with an elastic IP address in a public subnet. Another way is to use a public subnet and assign a public IP address to your task. We provide both examples below.

Example using a private subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service
 --task-definition sample-fargate-windows:1 --desired-count 1 --launch-type
 "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-
abcd1234],securityGroups=[sg-abcd1234]}"
```

Example using a public subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service
 --task-definition sample-fargate-windows:1 --desired-count 1 --launch-type
 "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-
abcd1234],securityGroups=[sg-abcd1234],assignPublicIp=ENABLED}"
```

The **create-service** command returns a description of the task definition after it completes its registration.

Step 5: List services

List the services for your cluster. You should see the service that you created in the previous section. You can take the service name or the full ARN that is returned from this command and use it to describe the service later.

```
aws ecs list-services --cluster fargate-cluster
```

Output:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-service"
    ]
}
```

Step 6: Describe the Running Service

Describe the service using the service name retrieved earlier to get more information about the task.

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

If successful, this will return a description of the service failures and services. For example, in services section, you will find information on deployments, such as the status of the tasks as running or pending. You may also find information on the task definition, the network configuration and time-stamped events. In the failures section, you will find information on failures, if any, associated with the call. For troubleshooting, see Service Event Messages. For more information about the service description, see Describe Services.

```
{
    "services": [
        {
            "status": "ACTIVE",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate-windows:1",
            "pendingCount": 2,
```

```
            "launchType": "FARGATE",
            "loadBalancers": [],
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "placementConstraints": [],
            "createdAt": 1510811361.128,
            "desiredCount": 2,
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "DISABLED"
                }
            },
            "platformVersion": "LATEST",
            "serviceName": "fargate-service",
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "deployments": [
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
                            ],
                            "assignPublicIp": "DISABLED"
                        }
                    },
                    "pendingCount": 2,
                    "launchType": "FARGATE",
                    "createdAt": 1510811361.128,
                    "desiredCount": 2,
```

```
                    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-
definition/sample-fargate-windows:1",
                    "updatedAt": 1510811361.128,
                    "platformVersion": "0.0.1",
                    "id": "ecs-svc/9223370526043414679",
                    "runningCount": 0
                }
            ],
            "events": [
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                    "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                    "createdAt": 1510811841.408
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                    "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                    "createdAt": 1510811601.938
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                    "createdAt": 1510811364.691
                }
            ],
            "runningCount": 0,
            "placementStrategy": []
        }
    ],
    "failures": []
}
```


Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

Creating an Amazon ECS task for the EC2 launch type with the AWS CLI

The following steps help you set up a cluster, register a task definition, run a task, and perform other common scenarios in Amazon ECS with the AWS CLI. Use the latest version of the AWS CLI. For more information on how to upgrade to the latest version, see Installing or updating to the latest version of the AWS CLI.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

Topics

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see Installing or updating to the latest version of the AWS CLI.
- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- You have a container instance IAM role created to use. For more informaton, see Amazon ECS container instance IAM role.
- You have a VPC created to use. For more information, see the section called “Create a virtual private cloud”.
- (Optional) AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see What is AWS CloudShell? in the *AWS CloudShell User Guide*.

Create a cluster

By default, your account receives a `default` cluster when you launch your first container instance.

> **Note**
>
> The benefit of using the `default` cluster that is provided for you is that you don't have to specify the `--cluster cluster_name` option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster cluster_name` for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name MyCluster
```

Output:

```
{
    "cluster": {
        "clusterName": "MyCluster",
        "status": "ACTIVE",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/MyCluster"
    }
}
```

Launch a container instance with the Amazon ECS AMI

Container instances are EC2 instances that run the Amazon ECS container agent and have been registered into a cluster. In this section, you'll launch an EC2 instance using the ECS-optimized AMI.

To launch a container instance with the AWS CLI

1. Retrieve the latest ECS-optimized Amazon Linux 2 AMI ID for your AWS Region using the following command. This command uses AWS Systems Manager Parameter Store to get the latest ECS-optimized AMI ID. The AMI includes the Amazon ECS container agent and Docker runtime pre-installed.

   ```
   aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/
   recommended --query 'Parameters[0].Value' --output text | jq -r '.image_id'
   ```

   Output:

   ```
   ami-abcd1234
   ```

2. Create a security group that allows SSH access for managing your container instance and HTTP access for the web server.

   ```
   aws ec2 create-security-group --group-name ecs-tutorial-sg --description "ECS
    tutorial security group"
   ```

   Output:

   ```
   {
       "GroupId": "sg-abcd1234"
   }
   ```

3. Add an inbound rule to the security group by running the following command.

```
aws ec2 authorize-security-group-ingress --group-id sg-abcd1234 --protocol tcp --
port 80 --cidr 0.0.0.0/0
```

Output:

```
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-efgh5678",
            "GroupId": "sg-abcd1234",
            "GroupOwnerId": "123456789012",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}
```

The security group now allows SSH access from the specified IP range and HTTP access from anywhere. In a production environment, you should restrict SSH access to your specific IP address and consider limiting HTTP access as needed.

4. Create an EC2 key pair for SSH access to your container instance.

```
aws ec2 create-key-pair --key-name ecs-tutorial-key --query 'KeyMaterial' --output
 text > ecs-tutorial-key.pem
chmod 400 ecs-tutorial-key.pem
```

The private key is saved to your local machine with appropriate permissions for SSH access.

5. Launch an EC2 instance using the ECS-optimized AMI and configure it to join your cluster.

```
aws ec2 run-instances --image-id ami-abcd1234 --instance-type t3.micro --key-
name ecs-tutorial-key --security-group-ids sg-abcd1234 --iam-instance-profile
 Name=ecsInstanceRole --user-data '#!/bin/bash
echo ECS_CLUSTER=MyCluster >> /etc/ecs/ecs.config'
{
    "Instances": [
```

```
        {
            "InstanceId": "i-abcd1234",
            "ImageId": "ami-abcd1234",
            "State": {
                "Code": 0,
                "Name": "pending"
            },
            "PrivateDnsName": "",
            "PublicDnsName": "",
            "StateReason": {
                "Code": "pending",
                "Message": "pending"
            },
            "InstanceType": "t3.micro",
            "KeyName": "ecs-tutorial-key",
            "LaunchTime": "2025-01-13T10:30:00.000Z"
        }
    ]
}
```

The user data script configures the Amazon ECS agent to register the instance with your `MyCluster`. The instance uses the `ecsInstanceRole` IAM role, which provides the necessary permissions for the agent.

List container instances

Within a few minutes of launching your container instance, the Amazon ECS agent registers the instance with your MyCluster cluster. You can list the container instances in a cluster by running the following command:

```
aws ecs list-container-instances --cluster MyCluster
```

Output:

```
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID"
    ]
}
```

Describe your container instance

After you have the ARN or ID of a container instance, you can use the **describe-container-instances** command to get valuable information on the instance, such as remaining and registered CPU and memory resources.

```
aws ecs describe-container-instances --cluster MyCluster --container-instances container_instance_ID
```

Output:

```
{
    "failures": [],
    "containerInstances": [
        {
            "status": "ACTIVE",
            "registeredResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
```

```
                    "integerValue": 0
                },
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "ec2InstanceId": "instance_id",
            "agentConnected": true,
            "containerInstanceArn": "arn:aws:ecs:us-west-2:aws_account_id:container-
instance/container_instance_ID",
            "pendingTasksCount": 0,
            "remainingResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                },
```

```
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "runningTasksCount": 0,
            "attributes": [
                {
                    "name": "com.amazonaws.ecs.capability.privileged-container"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
                },
                {
                    "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
                },
                {
                    "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
                }
            ],
            "versionInfo": {
                "agentVersion": "1.5.0",
                "agentHash": "b197edd",
                "dockerVersion": "DockerVersion: 1.7.1"
            }
        }
    ]
}
```


You can also find the Amazon EC2 instance ID that you can use to monitor the instance in the Amazon EC2 console or with the **aws ec2 describe-instances --instance-id *instance_id*** command.

Register a task definition

Before you can run a task on your Amazon ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses an `nginx` image. For more information about the available task definition parameters, see Amazon ECS task definitions.

```
{
    "family": "nginx-task",
    "containerDefinitions": [
        {
            "name": "nginx",
            "image": "public.ecr.aws/ecs-sample-image/amazon-ecs-sample:latest",
            "cpu": 256,
            "memory": 512,
            "essential": true,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ]
        }
    ],
    "requiresCompatibilities": ["EC2"],
    "networkMode": "bridge"
}
```

The above example JSON can be passed to the AWS CLI in two ways: You can save the task definition JSON as a file and pass it with the `--cli-input-json file://path_to_file.json` option. Or, you can escape the quotation marks in the JSON and pass the JSON container definitions on the command line. If you choose to pass the container definitions on the command line, your command additionally requires a `--family` parameter that is used to keep multiple versions of your task definition associated with each other.

To use a JSON file for container definitions:

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/nginx.json
```

The **register-task-definition** returns a description of the task definition after it completes its registration.

```
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/nginx-
task:1",
        "family": "nginx-task",
        "revision": 1,
        "status": "ACTIVE",
        "containerDefinitions": [
            {
                "name": "nginx",
                "image": "public.ecr.aws/docker/library/nginx:latest",
                "cpu": 256,
                "memory": 512,
                "essential": true,
                "portMappings": [
                    {
                        "containerPort": 80,
                        "hostPort": 80,
                        "protocol": "tcp"
                    }
                ],
                "environment": [],
                "mountPoints": [],
                "volumesFrom": []
            }
        ],
        "volumes": [],
        "networkMode": "bridge",
        "compatibilities": [
            "EC2"
        ],
        "requiresCompatibilities": [
            "EC2"
        ]
    }
}
```

List task definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the `family` and `revision` values that you can use together when calling **create-service**.

```
aws ecs list-task-definitions
```

Output:

```
{
    "taskDefinitionArns": [
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:2",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/nginx-task:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:3",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:4",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:5",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
    ]
}
```

Create a service

After you have registered a task for your account and have launched a container instance that is registered to your cluster, you can create an Amazon ECS service that runs and maintains a desired number of tasks simultaneously using the task definition that you registered. For this example, you place a single instance of the `nginx:1` task definition in your MyCluster cluster.

```
aws ecs create-service --cluster MyCluster --service-name nginx-service --task-
definition nginx-task:1 --desired-count 1
```

Output:

```
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:aws_account_id:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
```

```
        "taskDefinition": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/nginx-
task:1",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "EC2",
        "status": "ACTIVE",
        "createdAt": "2025-01-13T10:45:00.000Z"
    }
}
```

List services

List the services for your cluster. You should see the service that you created in the previous section. You can note the service ID or the full ARN that is returned from this command and use it to describe the service later.

```
aws ecs list-services --cluster MyCluster
```

Output:

```
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID"
    ]
}
```

Describe the service

Describe the service using the following command to get more information about the service.

```
aws ecs describe-services --cluster MyCluster --services nginx-service
```

Output:

```
{
    "services": [
        {
            "serviceArn": "arn:aws:ecs:us-east-1:aws_account_id:service/MyCluster/
nginx-service",
            "serviceName": "nginx-service",
```

```
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
            "taskDefinition": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
nginx-task:1",
            "desiredCount": 1,
            "runningCount": 1,
            "pendingCount": 0,
            "launchType": "EC2",
            "status": "ACTIVE",
            "createdAt": "2025-01-13T10:45:00.000Z",
            "events": [
                {
                    "id": "abcd1234-5678-90ab-cdef-1234567890ab",
                    "createdAt": "2025-01-13T10:45:30.000Z",
                    "message": "(service nginx-service) has started 1 tasks: (task
 abcd1234-5678-90ab-cdef-1234567890ab)."
                }
            ]
        }
    ]
}
```

Describe the running task

After describing the service, run the following command to get more information about the task that is running as part of your service.

```
aws ecs list-tasks --cluster MyCluster --service-name nginx-service
```

Output:

```
{
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/MyCluster/
abcd1234-5678-90ab-cdef-1234567890ab",
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/MyCluster",
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
nginx-task:1",
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-
instance/MyCluster/abcd1234-5678-90ab-cdef-1234567890ab",
            "lastStatus": "RUNNING",
            "desiredStatus": "RUNNING",
```

```
            "containers": [
                {
                    "containerArn": "arn:aws:ecs:us-east-1:aws_account_id:container/
MyCluster/abcd1234-5678-90ab-cdef-1234567890ab/abcd1234-5678-90ab-cdef-1234567890ab",
                    "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/MyCluster/
abcd1234-5678-90ab-cdef-1234567890ab",
                    "name": "nginx",
                    "lastStatus": "RUNNING",
                    "networkBindings": [
                        {
                            "bindIP": "0.0.0.0",
                            "containerPort": 80,
                            "hostPort": 80,
                            "protocol": "tcp"
                        }
                    ]
                }
            ],
            "createdAt": "2025-01-13T10:45:00.000Z",
            "startedAt": "2025-01-13T10:45:30.000Z"
        }
    ]
}
```

Test the web server

To test the web server

1. Retrieve the public IP address of your container instance by running the following command.

```
aws ec2 describe-instances --instance-ids i-abcd1234 --query
 'Reservations[0].Instances[0].PublicIpAddress' --output text
```

   Output:

```
203.0.113.25
```

2. After retrieving the IP address, run the following `curl` command with the IP address.

```
curl http://203.0.113.25
```

Output:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you can see this page, the nginx web server is successfully installed and
 working.</p>
...
</body>
</html>
```

The nginx welcome page confirms that your service is running successfully and accessible from the internet.

Clean up resources

To avoid incurring charges, clean up the resources that you created in this tutorial.

To clean up resources

1. Update the service to have zero desired tasks, then delete the service.

```
aws ecs update-service --cluster MyCluster --service nginx-service --desired-count
 0
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:123456789012:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "desiredCount": 0,
        "runningCount": 1,
        "pendingCount": 0,
        "status": "ACTIVE"
    }
}
```

2. Wait for the running tasks to stop, then delete the service.

```
aws ecs delete-service --cluster MyCluster --service nginx-service
{
    "service": {
        "serviceArn": "arn:aws:ecs:us-east-1:123456789012:service/MyCluster/nginx-
service",
        "serviceName": "nginx-service",
        "status": "DRAINING"
    }
}
```

3. Terminate the container instance you created.

```
aws ec2 terminate-instances --instance-ids i-abcd1234
{
    "TerminatingInstances": [
        {
            "InstanceId": "i-abcd1234",
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

4. Clean up the security group and key pair that you created.

```
aws ec2 delete-security-group --group-id sg-abcd1234
aws ec2 delete-key-pair --key-name ecs-tutorial-key
rm ecs-tutorial-key.pem
```

5. Delete the Amazon ECS cluster.

```
aws ecs delete-cluster --cluster MyCluster
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-east-1:123456789012:cluster/MyCluster",
```

```
        "clusterName": "MyCluster",
        "status": "INACTIVE"
    }
}
```

Configuring Amazon ECS to listen for CloudWatch Events events

Learn how to set up a simple Lambda function that listens for task events and writes them out to a CloudWatch Logs log stream.

Prerequisite: Set up a test cluster

If you do not have a running cluster to capture events from, follow the steps in the section called “Creating a cluster for the Fargate launch type” to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Lambda function correctly.

Step 1: Create the Lambda function

In this procedure, you create a simple Lambda function to serve as a target for Amazon ECS event stream messages.

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. On the **Author from scratch** screen, do the following:
    a. For **Name**, enter a value.
    b. For **Runtime**, choose your version of Python, for example, **Python 3.9**.
    c. For **Role**, choose **Create a new role with basic Lambda permissions**.
4. Choose **Create function**.
5. In the **Function code** section, edit the sample code to match the following example:

```
import json

def lambda_handler(event, context):
    if event["source"] != "aws.ecs":
```

```
        raise ValueError("Function only supports input from events with a source
 type of: aws.ecs")

    print('Here is the event:')
    print(json.dumps(event))
```

This is a simple Python 3.9 function that prints the event sent by Amazon ECS. If everything is configured correctly, at the end of this tutorial, you see that the event details appear in the CloudWatch Logs log stream associated with this Lambda function.

6. Choose **Save**.

Step 2: Register an event rule

Next, you create a CloudWatch Events event rule that captures task events coming from your Amazon ECS clusters. This rule captures all events coming from all clusters within the account where it is defined. The task messages themselves contain information about the event source, including the cluster on which it resides, that you can use to filter and sort events programmatically.

> **Note**
>
> When you use the AWS Management Console to create an event rule, the console automatically adds the IAM permissions necessary to grant CloudWatch Events permission to call your Lambda function. If you are creating an event rule using the AWS CLI, you need to grant this permission explicitly. For more information, see [Events in Amazon EventBridge](#) and [Amazon EventBridge event patterns](#) in the *Amazon EventBridge User Guide*.

To route events to your Lambda function

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation pane, choose **Events**, **Rules**, **Create rule**.
3. For **Event Source**, choose **ECS** as the event source. By default, the rule applies to all Amazon ECS events for all of your Amazon ECS groups. Alternatively, you can select specific events or a specific Amazon ECS group.
4. For **Targets**, choose **Add target**, for **Target type**, choose **Lambda function**, and then select your Lambda function.

5. Choose **Configure details**.
6. For **Rule definition**, type a name and description for your rule and choose **Create rule**.

Step 3: Create a task definition

Create a task definition.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task Definitions**.
3. Choose **Create new Task Definition**, **Create new revision with JSON**.
4. Copy and paste the following example task definition into the box and then choose **Save**.

```
{
   "containerDefinitions": [
      {
         "command": [
            "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</
title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
 Amazon ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html &&
 httpd-foreground\""
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
         "essential": true,
         "image": "public.ecr.aws/docker/library/httpd:2.4",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
         "name": "sample-fargate-app",
         "portMappings": [
            {
               "containerPort": 80,
```

```
                "hostPort": 80,
                "protocol": "tcp"
            }
        ]
    }
  ],
  "cpu": "256",
  "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
  "family": "fargate-task-definition",
  "memory": "512",
  "networkMode": "awsvpc",
  "runtimePlatform": {
        "operatingSystemFamily": "LINUX"
   },
  "requiresCompatibilities": [
      "FARGATE"
   ]
}
```

5. Choose **Create**.

Step 4: Test your rule

Finally, you create a CloudWatch Events event rule that captures task events coming from your Amazon ECS clusters. This rule captures all events coming from all clusters within the account where it is defined. The task messages themselves contain information about the event source, including the cluster on which it resides, that you can use to filter and sort events programmatically.

To test your rule

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. Choose **Task definitions**.
3. Choose **console-sample-app-static**, and then choose **Deploy**, **Run new task**.
4. For **Cluster**, choose default, and then choose **Deploy**.
5. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
6. On the navigation pane, choose **Logs** and select the log group for your Lambda function (for example, **/aws/lambda/***`my-function`*).
7. Select a log stream to view the event data.

Sending Amazon Simple Notification Service alerts for Amazon ECS task stopped events

Configure an Amazon EventBridge event rule that only captures task events where the task has stopped running because one of its essential containers has terminated. The event sends only task events with a specific `stoppedReason` property to the designated Amazon SNS topic.

Prerequisite: Set up a test cluster

If you do not have a running cluster to capture events from, follow the steps in Getting started with the console using Linux containers on AWS Fargate to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Amazon SNS topic and EventBridge rule correctly.

Prerequisite: Configure permissions for Amazon SNS

To allow EventBridge to publish to an Amazon SNS topic, use the aws sns get-topic-attributes and the aws sns set-topic-attributes commands.

For information about how to add the permission, see Amazon SNS permissions in the *Amazon Simple Notification Service Developer Guide*

Add the following permissions:

```
{
  "Sid": "PublishEventsToMyTopic",
  "Effect": "Allow",
  "Principal": {
     "Service": "events.amazonaws.com"
  },
  "Action": "sns: Publish",
  "Resource": "arn:aws:sns:region:account-id:TaskStoppedAlert",
}
```

Step 1: Create and subscribe to an Amazon SNS topic

For this tutorial, you configure an Amazon SNS topic to serve as an event target for your new event rule.

For information about how to create and subscribe to an Amazon SNS topic , see [Getting started with Amazon SNS]() in the *Amazon Simple Notification Service Developer Guide* and use the following table to determine what options to select.

Option	Value	
Type	Standard	
Name	TaskStoppedAlert	
Protocol	Email	
Endpoint	An email address to which you currently have access	

Step 2: Register an event rule

Next, you register an event rule that captures only task-stopped events for tasks with stopped containers.

For information about how to create and subscribe to an Amazon SNS topic , see [Create a rule in Amazon EventBridge]() in the *Amazon EventBridge User Guide* and use the following table to determine what options to select.

Option	Value	
Rule type	Rule with an event pattern	
Event source	AWS events or EventBridge partner events	
Event pattern	Custom pattern (JSON editor)	

Event pattern	`{`
	`   "source":[`
	`      "aws.ecs"`
	`   ],`
	`   "detail-type":[`	

Option	Value
	`"ECS Task State Change" ], "detail":{ "lastStatus":[ "STOPPED" ], "stoppedReason":[ "Essential container in task exited" ] } }`
Target type	AWS service
Target	SNS topic
Topic	TaskStoppedAlert (The topic you created in Step 1)

Step 3: Test your rule

Verify that the rule is working by running a task that exits shortly after it starts. If your event rule is configured correctly, you receive an email message within a few minutes with the event text. If you have an existing task definition that can satisfy the rule requirements, run a task using it. If you do not, the following steps will walk you through registering a Fargate task definition and running it that will.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**.
3. Choose **Create new task definition**, **Create new task definition with JSON**.
4. In the JSON editor box, edit your JSON file, copy the following into the editor.

```
{
   "containerDefinitions":[
```

```
      {
         "command":[
            "sh",
            "-c",
            "sleep 5"
         ],
         "essential":true,
         "image":"public.ecr.aws/amazonlinux/amazonlinux:latest",
         "name":"test-sleep"
      }
   ],
   "cpu":"256",
   "executionRoleArn":"arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family":"fargate-task-definition",
   "memory":"512",
   "networkMode":"awsvpc",
   "requiresCompatibilities":[
      "FARGATE"
   ]
}
```

5. Choose **Create**.

To run a task from the console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. On the **Clusters** page, choose the cluster you created in the prerequisites.
3. From the **Tasks** tab, choose **Run new task**.
4. For **Application type**, choose **Task**.
5. For **Task definition**, choose **fargate-task-definition**.
6. For **Desired tasks**, enter the number of tasks to launch.
7. Choose **Create**.

Concatenating multiline or stack-trace Amazon ECS log messages

Beginning with AWS for Fluent Bit version 2.22.0, a multiline filter is included. The multiline filter helps concatenate log messages that originally belong to one context but were split across multiple records or log lines. For more information about the multiline filter, see the Fluent Bit documentation.

Common examples of split log messages are:

- Stack traces.
- Applications that print logs on multiple lines.
- Log messages that were split because they were longer than the specified runtime max buffer size. You can concatenate log messages split by the container runtime by following the example on GitHub: FireLens Example: Concatenate Partial/Split Container Logs.

Required IAM permissions

You have the necessary IAM permissions for the container agent to pull the container images from Amazon ECR and for the container to route logs to CloudWatch Logs.

For these permissions, you must have the following roles:

- A task IAM role.
- A task execution IAM role.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
```

```
   "Action": [
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:PutLogEvents"
   ],
   "Resource": "*"
  }]
}
```

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
8. Choose **Create policy** to save your new policy.

Determine when to use the multiline log setting

The following are example log snippets that you see in the CloudWatch Logs console with the default log setting. You can look at the line that starts with `log` to determine if you need the multiline filter. When the context is the same, you can use the multiline log setting, In this example, the context is "com.myproject.model.MyProject".

```
2022-09-20T15:47:56:595-05-00                           {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-
app", "source=": "stdout", "log": ": "     at com.myproject.modele.
(MyProject.badMethod.java:22)",
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log": ": "     at com.myproject.model.MyProject.badMethod(MyProject.java:22)",
      "ecs_cluster": "default",
```

```
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE",
      "ecs_task_definition": "firelense-example-multiline:3"
     }
```

```
2022-09-20T15:47:56:595-05-00                           {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app", "stdout",
 "log": ": "     at com.myproject.modele.(MyProject.oneMoreMethod.java:18)",
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log": ": "     at
 com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE,
      "ecs_task_definition": "firelense-example-multiline:3"
     }
```

After you use the multiline log setting, the output will look similar to the example below.

```
2022-09-20T15:47:56:595-05-00                           {"container_id":
 "82ba37cada1d44d389b03e78caf74faa-EXAMPLE", "container_name": "example-app",
 "stdout",...
    {
      "container_id":  "82ba37cada1d44d389b03e78caf74faa-EXAMPLE",
      "container_name: ": "example-app",
      "source": "stdout",
      "log:    "September 20, 2022 06:41:48 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!\n
    at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at
    at com.myproject.model.MyProject.oneMoreMethod(MyProject.java:18)
 com.myproject.module.MyProject.main(MyProject.java:6)",
      "ecs_cluster": "default",
      "ecs_task_arn": "arn:aws:region:123456789012:task/default/
b23c940d29ed4714971cba72cEXAMPLE",
      "ecs_task_definition": "firelense-example-multiline:2"
     }
```

Parse and concatenate options

To parse logs and concatenate lines that were split because of newlines, you can use either of these two options.

- Use your own parser file that contains the rules to parse and concatenate lines that belong to the same message.
- Use a Fluent Bit built-in parser. For a list of languages supported by the Fluent Bit built-in parsers, see Fluent Bit documentation.

The following tutorial walks you through the steps for each use case. The steps show you how to concatenate multilines and send the logs to Amazon CloudWatch. You can specify a different destination for your logs.

Example: Use a parser that you create

In this example, you will complete the following steps:

1. Build and upload the image for a Fluent Bit container.
2. Build and upload the image for a demo multiline application that runs, fails, and generates a multiline stack trace.
3. Create the task definition and run the task.
4. View the logs to verify that messages that span multiple lines appear concatenated.

Build and upload the image for a Fluent Bit container

This image will include the parser file where you specify the regular expression and a configuration file that references the parser file.

1. Create a folder with the name `FluentBitDockerImage`.
2. Within the folder, create a parser file that contains the rules to parse the log and concatenate lines that belong in the same message.

   a. Paste the following contents in the parser file:

```
[MULTILINE_PARSER]
    name          multiline-regex-test
    type          regex
```

```
    flush_timeout 1000
    #
    # Regex rules for multiline parsing
    # ---------------------------------
    #
    # configuration hints:
    #
    #  - first state always has the name: start_state
    #  - every field in the rule must be inside double quotes
    #
    # rules |   state name  | regex pattern                  | next state
    # ------|---------------|--------------------------------------------
    rule      "start_state"   "/(Dec \d+ \d+\:\d+\:\d+)(.*)/"  "cont"
    rule      "cont"          "/^\s+at.*/"                     "cont"
```

As you customize your regex pattern, we recommend you use a regular expression editor to test the expression.

b. Save the file as `parsers_multiline.conf`.

3. Within the `FluentBitDockerImage` folder, create a custom configuration file that references the parser file that you created in the previous step.

   For more information about the custom configuration file, see [Specifying a custom configuration file]() in the *Amazon Elastic Container Service Developer Guide*

   a. Paste the following contents in the file:

```
[SERVICE]
    flush                 1
    log_level             info
    parsers_file          /parsers_multiline.conf

[FILTER]
    name                  multiline
    match                 *
    multiline.key_content log
    multiline.parser      multiline-regex-test
```

> **Note**
>
> You must use the absolute path of the parser.

b. Save the file as `extra.conf`.

4. Within the `FluentBitDockerImage` folder, create the Dockerfile with the Fluent Bit image and the parser and configuration files that you created.

   a. Paste the following contents in the file:

```
FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest

ADD parsers_multiline.conf /parsers_multiline.conf
ADD extra.conf /extra.conf
```

   b. Save the file as `Dockerfile`.

5. Using the Dockerfile, build a custom Fluent Bit image with the parser and custom configuration files included.

> **Note**
>
> You can place the parser file and configuration file anywhere in the Docker image except `/fluent-bit/etc/fluent-bit.conf` as this file path is used by FireLens.

   a. Build the image: `docker build -t fluent-bit-multiline-image.`

      Where: `fluent-bit-multiline-image` is the name for the image in this example.

   b. Verify that the image was created correctly: `docker images —filter reference=fluent-bit-multiline-image`

      If successful, the output shows the image and the `latest` tag.

6. Upload the custom Fluent Bit image to Amazon Elastic Container Registry.

   a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

      Where: `fluent-bit-multiline-repo` is the name for the repository and `us-east-1` is the region in this example.

      The output gives you the details of the new repository.

b. Tag your image with the `repositoryUri` value from the previous output: `docker tag` *`fluent-bit-multiline-image repositoryUri`*

   Example: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

c. Run the docker image to verify it ran correctly: `docker images —filter reference=`*`repositoryUri`*

   In the output, the repository name changes from fluent-bit-multiline-repo to the `repositoryUri`.

d. Authenticate to Amazon ECR by running the `aws ecr get-login-password` command and specifying the registry ID you want to authenticate to: `aws ecr get-login-password | docker login --username AWS --password-stdin` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com`

   Example: `ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

   A successful login message appears.

e. Push the image to Amazon ECR: `docker push` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository name`*

   Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

Build and upload the image for a demo multiline application

This image will include a Python script file that runs the application and a sample log file.

When you run the task, the application simulates runs, then fails and creates a stack trace.

1. Create a folder named `multiline-app`: `mkdir multiline-app`
2. Create a Python script file.

   a. Within the `multiline-app` folder, create a file and name it `main.py`.

   b. Paste the following contents in the file:

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0

for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")
```

c. Save the `main.py` file.

3. Create a sample log file.

   a. Within the `multiline-app` folder, create a file and name it `test.log`.

   b. Paste the following contents in the file:

```
single line...
Dec 14 06:41:08 Exception in thread "main" java.lang.RuntimeException:
 Something has gone wrong, aborting!
    at com.myproject.module.MyProject.badMethod(MyProject.java:22)
    at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)
    at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)
    at com.myproject.module.MyProject.someMethod(MyProject.java:10)
    at com.myproject.module.MyProject.main(MyProject.java:6)
another line...
```

   c. Save the `test.log` file.

4. Within the `multiline-app` folder, create the Dockerfile.

   a. Paste the following contents in the file:

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
```

```
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3

WORKDIR /usr/local/bin

COPY main.py .

CMD ["python3", "main.py"]
```

   b. Save the `Dockerfile` file.

5. Using the Dockerfile, build an image.

   a. Build the image: `docker build -t multiline-app-image`

      Where: `multiline-app-image` is the name for the image in this example.

   b. Verify that the image was created correctly: `docker images —filter reference=multiline-app-image`

      If successful, the output shows the image and the `latest` tag.

6. Upload the image to Amazon Elastic Container Registry.

   a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

      Where: `multiline-app-repo` is the name for the repository and `us-east-1` is the region in this example.

      The output gives you the details of the new repository. Note the `repositoryUri` value as you will need it in the next steps.

   b. Tag your image with the `repositoryUri` value from the previous output: `docker tag` *`multiline-app-image repositoryUri`*

      Example: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

   c. Run the docker image to verify it ran correctly: `docker images —filter reference=`*`repositoryUri`*

      In the output, the repository name changes from `multiline-app-repo` to the `repositoryUri` value.

d. Push the image to Amazon ECR: `docker push aws_account_id.dkr.ecr.region.amazonaws.com/repository name`

   Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

Create the task definition and run the task

1. Create a task definition file with the file name `multiline-task-definition.json`.
2. Paste the following contents in the `multiline-task-definition.json` file:

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN,
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-
multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-
image:latest",
            "name": "app",
            "logConfiguration": {
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "cloudwatch_logs",
                    "region": "us-east-1",
                    "log_group_name": "multiline-test/application",
```

```
                    "auto_create_group": "true",
                    "log_stream_prefix": "multiline-"
                }
            },
            "memoryReservation": 100
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}
```

Replace the following in the `multiline-task-definition.json` task definition:

a. *`task role ARN`*

   To find the task role ARN, go to the IAM console. Choose **Roles** and find the `ecs-task-role-for-firelens` task role that you created. Choose the role and copy the **ARN** that appears in the **Summary** section.

b. *`execution role ARN`*

   To find the execution role ARN, go to the IAM console. Choose **Roles** and find the `ecsTaskExecutionRole` role. Choose the role and copy the **ARN** that appears in the **Summary** section.

c. *`aws_account_id`*

   To find your `aws_account_id`, log into the AWS Management Console. Choose your user name on the top right and copy your Account ID.

d. *`us-east-1`*

   Replace the region if necessary.

3. Register the task definition file: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region` *`region`*
4. Open the console at https://console.aws.amazon.com/ecs/v2.
5. In the navigation pane, choose **Task Definitions** and then choose the `firelens-example-multiline` family because we registered the task definition to this family in the first line of the task definition above.

6. Choose the latest version.
7. Choose the **Deploy**, **Run task**.
8. On the **Run Task** page, For **Cluster**, choose the cluster, and then under **Networking**, for **Subnets**, choose the available subnets for your task.
9. Choose **Create**.

Verify that multiline log messages in Amazon CloudWatch appear concatenated

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. From the navigation pane, expand **Logs** and choose **Log groups**.
3. Choose the `multiline-test/applicatio` log group.
4. Choose the log. View messages. Lines that matched the rules in the parser file are concatenated and appear as a single message.

   The following log snippet shows lines concatenated in a single Java stack trace event:

```
{
    "container_id": "xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "log": "Dec 14 06:41:08 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!\n
 at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at
 com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)\n
 at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)\n
 at com.myproject.module.MyProject.someMethod(MyProject.java:10)\n    at
 com.myproject.module.MyProject.main(MyProject.java:6)",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:2"
}
```

   The following log snippet shows how the same message appears with just a single line if you run an Amazon ECS container that is not configured to concatenate multiline log messages.

```
{
    "log": "Dec 14 06:41:08 Exception in thread \"main\"
 java.lang.RuntimeException: Something has gone wrong, aborting!",
    "container_id": "xxxxxx-xxxxxx",
```

```
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
}
```

Example: Use a Fluent Bit built-in parser

In this example, you will complete the following steps:

1. Build and upload the image for a Fluent Bit container.
2. Build and upload the image for a demo multiline application that runs, fails, and generates a multiline stack trace.
3. Create the task definition and run the task.
4. View the logs to verify that messages that span multiple lines appear concatenated.

Build and upload the image for a Fluent Bit container

This image will include a configuration file that references the Fluent Bit parser.

1. Create a folder with the name `FluentBitDockerImage`.
2. Within the `FluentBitDockerImage` folder, create a custom configuration file that references the Fluent Bit built-in parser file.

   For more information about the custom configuration file, see [Specifying a custom configuration file]() in the *Amazon Elastic Container Service Developer Guide*

   a. Paste the following contents in the file:

   ```
   [FILTER]
       name                  multiline
       match                 *
       multiline.key_content log
       multiline.parser      go
   ```

   b. Save the file as `extra.conf`.
3. Within the `FluentBitDockerImage` folder, create the Dockerfile with the Fluent Bit image and the parser and configuration files that you created.

a. Paste the following contents in the file:

```
FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest
ADD extra.conf /extra.conf
```

b. Save the file as `Dockerfile`.

4. Using the Dockerfile, build a custom Fluent Bit image with the custom configuration file included.

> **Note**
>
> You can place the configuration file anywhere in the Docker image except `/fluent-bit/etc/fluent-bit.conf` as this file path is used by FireLens.

a. Build the image: `docker build -t fluent-bit-multiline-image.`

   Where: `fluent-bit-multiline-image` is the name for the image in this example.

b. Verify that the image was created correctly: `docker images —filter reference=fluent-bit-multiline-image`

   If successful, the output shows the image and the `latest` tag.

5. Upload the custom Fluent Bit image to Amazon Elastic Container Registry.

a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

   Where: `fluent-bit-multiline-repo` is the name for the repository and `us-east-1` is the region in this example.

   The output gives you the details of the new repository.

b. Tag your image with the `repositoryUri` value from the previous output: `docker tag` *`fluent-bit-multiline-image repositoryUri`*

   Example: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

c. Run the docker image to verify it ran correctly: `docker images —filter reference=`*`repositoryUri`*

   In the output, the repository name changes from fluent-bit-multiline-repo to the `repositoryUri`.

d. Authenticate to Amazon ECR by running the `aws ecr get-login-password` command and specifying the registry ID you want to authenticate to: `aws ecr get-login-password | docker login --username AWS --password-stdin` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com`

   Example: `ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

   A successful login message appears.

e. Push the image to Amazon ECR: `docker push` *`registry ID`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository name`*

   Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

Build and upload the image for a demo multiline application

This image will include a Python script file that runs the application and a sample log file.

1. Create a folder named `multiline-app`: `mkdir multiline-app`
2. Create a Python script file.

   a. Within the `multiline-app` folder, create a file and name it `main.py`.

   b. Paste the following contents in the file:

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0

for i in range(10):
    print("app running normally...")
```

```
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")
```

c. Save the `main.py` file.

3. Create a sample log file.

   a. Within the `multiline-app` folder, create a file and name it `test.log`.

   b. Paste the following contents in the file:

```
panic: my panic

goroutine 4 [running]:
panic(0x45cb40, 0x47ad70)
  /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8 sp=0xc42003f710
 pc=0x422f7c
main.main.func1(0xc420024120)
  foo.go:6 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0
 sp=0xc42003f7d8 pc=0x44b4d1
created by main.main
  foo.go:5 +0x58

goroutine 1 [chan receive]:
runtime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00
 pc=0x42503c
runtime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
 pc=0x42512e
runtime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)
  /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
 pc=0x4046b4
runtime.chanrecv1(0xc420024120, 0x0)
  /usr/local/go/src/runtime/chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20
 pc=0x40439b
```

```
main.main()
  foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef
runtime.main()
  /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
 pc=0x424bad
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc420053fe8
 sp=0xc420053fe0 pc=0x44b4d1

goroutine 2 [force gc (idle)]:
runtime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
 pc=0x42503c
runtime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
 pc=0x42512e
runtime.forcegchelper()
  /usr/local/go/src/runtime/proc.go:238 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8
 pc=0x424e5c
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8
 sp=0xc42003e7e0 pc=0x44b4d1
created by runtime.init.4
  /usr/local/go/src/runtime/proc.go:227 +0x35

goroutine 3 [GC sweep wait]:
runtime.gopark(0x4739b8, 0x4ad7e0, 0x46fdd2, 0xd, 0x419914, 0x1)
  /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003ef60 sp=0xc42003ef30
 pc=0x42503c
runtime.goparkunlock(0x4ad7e0, 0x46fdd2, 0xd, 0x14, 0x1)
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003efa0 sp=0xc42003ef60
 pc=0x42512e
runtime.bgsweep(0xc42001e150)
  /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8
 sp=0xc42003efa0 pc=0x419973
runtime.goexit()
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003efe0
 sp=0xc42003efd8 pc=0x44b4d1
created by runtime.gcenable
  /usr/local/go/src/runtime/mgc.go:216 +0x58
one more line, no multiline
```

c. Save the `test.log` file.

4. Within the `multiline-app` folder, create the Dockerfile.

   a. Paste the following contents in the file:

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:latest
   ADD test.log /test.log

   RUN yum upgrade -y && yum install -y python3

   WORKDIR /usr/local/bin

   COPY main.py .

   CMD ["python3", "main.py"]
   ```

   b. Save the `Dockerfile` file.

5. Using the Dockerfile, build an image.

   a. Build the image: `docker build -t multiline-app-image`

      Where: `multiline-app-image` is the name for the image in this example.

   b. Verify that the image was created correctly: `docker images —filter reference=multiline-app-image`

      If successful, the output shows the image and the `latest` tag.

6. Upload the image to Amazon Elastic Container Registry.

   a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

      Where: `multiline-app-repo` is the name for the repository and `us-east-1` is the region in this example.

      The output gives you the details of the new repository. Note the `repositoryUri` value as you will need it in the next steps.

   b. Tag your image with the `repositoryUri` value from the previous output: `docker tag` *`multiline-app-image repositoryUri`*

      Example: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

c. Run the docker image to verify it ran correctly: `docker images —filter reference=repositoryUri`

   In the output, the repository name changes from `multiline-app-repo` to the `repositoryUri` value.

d. Push the image to Amazon ECR: `docker push aws_account_id.dkr.ecr.region.amazonaws.com/repository name`

   Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

Create the task definition and run the task

1. Create a task definition file with the file name `multiline-task-definition.json`.
2. Paste the following contents in the `multiline-task-definition.json` file:

```
{
    "family": "firelens-example-multiline",
    "taskRoleArn": "task role ARN,
    "executionRoleArn": "execution role ARN",
    "containerDefinitions": [
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-image:latest",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit",
                "options": {
                    "config-file-type": "file",
                    "config-file-value": "/extra.conf"
                }
            },
            "memoryReservation": 50
        },
        {
            "essential": true,
            "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-image:latest",
            "name": "app",
            "logConfiguration": {
```

```
                "logDriver": "awsfirelens",
                "options": {
                    "Name": "cloudwatch_logs",
                    "region": "us-east-1",
                    "log_group_name": "multiline-test/application",
                    "auto_create_group": "true",
                    "log_stream_prefix": "multiline-"
                }
            },
            "memoryReservation": 100
        }
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512"
}
```

Replace the following in the `multiline-task-definition.json` task definition:

a. *`task role ARN`*

   To find the task role ARN, go to the IAM console. Choose **Roles** and find the `ecs-task-role-for-firelens` task role that you created. Choose the role and copy the **ARN** that appears in the **Summary** section.

b. *`execution role ARN`*

   To find the execution role ARN, go to the IAM console. Choose **Roles** and find the `ecsTaskExecutionRole` role. Choose the role and copy the **ARN** that appears in the **Summary** section.

c. *`aws_account_id`*

   To find your `aws_account_id`, log into the AWS Management Console. Choose your user name on the top right and copy your Account ID.

d. *`us-east-1`*

   Replace the region if necessary.

3. Register the task definition file: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region us-east-1`
4. Open the console at https://console.aws.amazon.com/ecs/v2.

5. In the navigation pane, choose **Task Definitions** and then choose the `firelens-example-multiline` family because we registered the task definition to this family in the first line of the task definition above.
6. Choose the latest version.
7. Choose the **Deploy**, **Run task**.
8. On the **Run Task** page, For **Cluster**, choose the cluster, and then under **Networking**, for **Subnets**, choose the available subnets for your task.
9. Choose **Create**.

Verify that multiline log messages in Amazon CloudWatch appear concatenated

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. From the navigation pane, expand **Logs** and choose **Log groups**.
3. Choose the `multiline-test/applicatio` log group.
4. Choose the log and view the messages. Lines that matched the rules in the parser file are concatenated and appear as a single message.

   The following log snippet shows a Go stack trace that is concatenated into a single event:

```
{
    "log": "panic: my panic\n\ngoroutine 4 [running]:\npanic(0x45cb40,
 0x47ad70)\n  /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8
 sp=0xc42003f710 pc=0x422f7c\nmain.main.func1(0xc420024120)\n  foo.go:6
 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339\nruntime.goexit()\n  /usr/
local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0 sp=0xc42003f7d8
 pc=0x44b4d1\ncreated by main.main\n  foo.go:5 +0x58\n\ngoroutine 1 [chan receive]:
\nruntime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)\n  /usr/
local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00 pc=0x42503c
\nruntime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)\n
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
 pc=0x42512e\nruntime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)\n
  /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
 pc=0x4046b4\nruntime.chanrecv1(0xc420024120, 0x0)\n  /usr/local/go/src/runtime/
chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20 pc=0x40439b\nmain.main()\n
 foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef\nruntime.main()\n
  /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
 pc=0x424bad\nruntime.goexit()\n  /usr/local/go/src/runtime/asm_amd64.s:2337
 +0x1 fp=0xc420053fe8 sp=0xc420053fe0 pc=0x44b4d1\n\ngoroutine 2 [force gc
 (idle)]:\nruntime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)\n  /
```

```
usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
 pc=0x42503c\nruntime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)\n
  /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
 pc=0x42512e\nruntime.forcegchelper()\n  /usr/local/go/src/runtime/proc.go:238
 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8 pc=0x424e5c\nruntime.goexit()\n  /usr/
local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8 sp=0xc42003e7e0
 pc=0x44b4d1\ncreated by runtime.init.4\n  /usr/local/go/src/runtime/proc.go:227
 +0x35\n\ngoroutine 3 [GC sweep wait]:\nruntime.gopark(0x4739b8, 0x4ad7e0,
 0x46fdd2, 0xd, 0x419914, 0x1)\n  /usr/local/go/src/runtime/proc.go:280 +0x12c
 fp=0xc42003ef60 sp=0xc42003ef30 pc=0x42503c\nruntime.goparkunlock(0x4ad7e0,
 0x46fdd2, 0xd, 0x14, 0x1)\n  /usr/local/go/src/runtime/proc.go:286 +0x5e
 fp=0xc42003efa0 sp=0xc42003ef60 pc=0x42512e\nruntime.bgsweep(0xc42001e150)\n
  /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8 sp=0xc42003efa0
 pc=0x419973\nruntime.goexit()\n  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1
 fp=0xc42003efe0 sp=0xc42003efd8 pc=0x44b4d1\ncreated by runtime.gcenable\n  /usr/
local/go/src/runtime/mgc.go:216 +0x58",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:2"
}
```

The following log snippet shows how the same event appears if you run an ECS container that is not configured to concatenate multiline log messages. The log field contains a single line.

```
{
    "log": "panic: my panic",
    "container_id": "xxxxxx-xxxxxx",
    "container_name": "app",
    "source": "stdout",
    "ecs_cluster": "default",
    "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
    "ecs_task_definition": "firelens-example-multiline:3"
```

Note

If your logs go to log files instead of the standard output, we recommend specifying the `multiline.parser` and `multiline.key_content` configuration parameters in the Tail input plugin instead of the Filter.

Deploying Fluent Bit on Amazon ECS Windows containers

Fluent Bit is a fast and flexible log processor and router supported by various operating systems. It can be used to route logs to various AWS destinations such as Amazon CloudWatch Logs, Firehose Amazon S3, and Amazon OpenSearch Service. Fluent Bit supports common partner solutions such as Datadog, Splunk, and custom HTTP servers. For more information about Fluent Bit, see the Fluent Bit website.

The **AWS for Fluent Bit** image is available on Amazon ECR on both the Amazon ECR Public Gallery and in an Amazon ECR repository in most Regions for high availability. For more information, see aws-for-fluent-bit on the GitHub website.

This tutorial walks you through how to deploy Fluent Bit containers on their Windows instances running in Amazon ECS to stream logs generated by the Windows tasks to Amazon CloudWatch for centralized logging.

This tutorial uses the following approach:

- Fluent Bit runs as a service with the Daemon scheduling strategy. This strategy ensures that a single instance of Fluent Bit always runs on the container instances in the cluster.
  - Listens on port 24224 using the forward input plug-in.
  - Expose port 24224 to the host so that the docker runtime can send logs to Fluent Bit using this exposed port.
  - Has a configuration which allows Fluent Bit to send the logs records to specified destinations.
- Launch all other Amazon ECS task containers using the fluentd logging driver. For more information, see Fluentd logging driver on the Docker documentation website.
  - Docker connects to the TCP socket 24224 on localhost inside the host namespace.
  - The Amazon ECS agent adds labels to the containers which includes the cluster name, task definition family name, task definition revision number, task ARN, and the container name. The same information is added to the log record using the labels option of the fluentd docker

logging driver. For more information, see [labels, labels-regex, env, and env-regex] on the Docker documentation website.

- Because the `async` option of the fluentd logging driver is set to `true`, when the Fluent Bit container is restarted, docker buffers the logs until the Fluent Bit container is restarted. You can increase the buffer limit by setting the fluentd-buffer-limit option. For more information, see [fluentd-buffer-limit] on the Docker documentation website.

The work flow is as follows:

- The Fluent Bit container starts and listens on port 24224 which is exposed to the host.
- Fluent Bit uses the task IAM role credentials specified in its task definition.
- Other tasks launched on the same instance use the fluentd docker logging driver to connect to the Fluent Bit container on port 24224.
- When the application containers generate logs, docker runtime tags those records, adds additional metadata specified in labels, and then forwards them on port 24224 in the host namespace.
- Fluent Bit receives the log record on port 24224 because it is exposed to the host namespace.
- Fluent Bit performs its internal processing and routes the logs as specified.

This tutorial uses the default CloudWatch Fluent Bit configuration which does the following:

- Creates a new log group for each cluster and task definition family.
- Creates a new log stream for each task container in above generated log group whenever a new task is launched. Each stream will be marked with the task id to which the container belongs.
- Adds additional metadata including the cluster name, task ARN, task container name, task definition family, and the task definition revision number in each log entry.

  For example, if you have `task_1` with `container_1` and `container_2` and `task_2` with `container_3`, then the following are the CloudWatch log streams:

  - `/aws/ecs/windows.ecs_task_1`

    `task-out.`*`TASK_ID`*`.container_1`

    `task-out.`*`TASK_ID`*`.container_2`
  - `/aws/ecs/windows.ecs_task_2`

```
task-out.TASK_ID.container_3
```

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

Steps

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The latest version of the AWS CLI is installed and configured. For more information, see Installing or updating to the latest version of the AWS CLI.
- The `aws-for-fluent-bit` container image is available for the following Windows operating systems:
  - Windows Server 2019 Core
  - Windows Server 2019 Full
  - Windows Server 2022 Core

  - Windows Server 2022 Full
- The steps in Set up to use Amazon ECS have been completed.
- You have a cluster. In this tutorial, the cluster name is **FluentBit-cluster**.
- You have a VPC with a public subnet where the EC2 instance will be launched. You can use your default VPC. You can also use a private subnet that allows Amazon CloudWatch endpoints to reach the subnet. For more information about Amazon CloudWatch endpoints, see Amazon CloudWatch endpoints and quotas in the *AWS General Reference*. For information about how to use the Amazon VPC wizard to create a VPC, see the section called “Create a virtual private cloud”.

Step 1: Create the IAM access roles

Create the Amazon ECS IAM roles.

1. Create the Amazon ECS container instance role named "ecsInstanceRole". For more information, see Amazon ECS container instance IAM role.
2. Create an IAM role for the Fluent Bit task named `fluentTaskRole`. For more information, see the section called “Task IAM role”.

   The IAM permissions granted in this IAM role are assumed by the task containers. In order to allow Fluent Bit to send logs to CloudWatch, you need to attach the following permissions to the task IAM role.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",
            "logs:CreateLogGroup",
            "logs:DescribeLogStreams",
            "logs:PutLogEvents"
        ],
        "Resource": "*"
    }
```

```
    ]
}
```

3. Attach the policy to the role.

   a. Save the above content in a file named `fluent-bit-policy.json`.

   b. Run the following command to attach the inline policy to `fluentTaskRole` IAM role.

```
aws iam put-role-policy --role-name fluentTaskRole --policy-name
 fluentTaskPolicy --policy-document file://fluent-bit-policy.json
```

Step 2: Create an Amazon ECS Windows container instance

Create an Amazon ECS Windows container instance.

To create an Amazon ECS instance

1. Use the `aws ssm get-parameters` command to retrieve the AMI ID for the Region that hosts your VPC. For more information, see Retrieving Amazon ECS-Optimized AMI metadata.
2. Use the Amazon EC2 console to launch the instance.

   a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

   b. From the navigation bar, select the Region to use.

   c. From the **EC2 Dashboard**, choose **Launch instance**.

   d. For **Name**, enter a unique name.

   e. For **Application and OS Images (Amazon Machine Image)**, choose the AMI that you retrieved in the first step.

   f. For **Instance type**, choose `t3.xlarge`.

   g. For **Key pair (login)**, choose a key pair.

   h. Under **Network settings**, for **Security group**, choose an existing security group, or create a new one.

   i. Under **Network settings**, for **Auto-assign Public IP**, select **Enable**.

   j. Under **Advanced details**, for **IAM instance profile** , choose **ecsInstanceRole**.

   k. Configure your Amazon ECS container instance with the following user data. Under **Advanced Details**, paste the following script into the **User data** field, replacing *`cluster_name`* with the name of your cluster.

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster cluster-name -EnableTaskENI -EnableTaskIAMRole -
LoggingDrivers '["awslogs","fluentd"]'
</powershell>
```

l. When you are ready, select the acknowledgment field, and then choose **Launch Instances**.

m. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.

Step 3: Configure Fluent Bit

You can use the following default configuration provided by AWS to get quickly started:

- Amazon CloudWatch which is based on the Fluent Bit plug-in for Amazon CloudWatch on the *Fluent Bit Official Manual*.

Alternatively, you can use other default configurations provided by AWS. For more information, see Overriding the entrypoint for the Windows image on the `aws-for-fluent-bit` the Github website.

The default Amazon CloudWatch Fluent Bit configuration is shown below.

Replace the following variables:

- *`region`* with the Region where you want to send the Amazon CloudWatch logs.

```
[SERVICE]
    Flush               5
    Log_Level           info
    Daemon              off

[INPUT]
    Name                forward
    Listen              0.0.0.0
    Port                24224
    Buffer_Chunk_Size   1M
    Buffer_Max_Size     6M
    Tag_Prefix          ecs.
```

```
# Amazon ECS agent adds the following log keys as labels to the docker container.
# We would use fluentd logging driver to add these to log record while sending it to
 Fluent Bit.
[FILTER]
    Name                modify
    Match               ecs.*
    Rename              com.amazonaws.ecs.cluster ecs_cluster
    Rename              com.amazonaws.ecs.container-name ecs_container_name
    Rename              com.amazonaws.ecs.task-arn ecs_task_arn
    Rename              com.amazonaws.ecs.task-definition-family
 ecs_task_definition_family
    Rename              com.amazonaws.ecs.task-definition-version
 ecs_task_definition_version

[FILTER]
    Name                rewrite_tag
    Match               ecs.*
    Rule                $ecs_task_arn ^([a-z-:0-9]+)/([a-zA-Z0-9-_]+)/([a-z0-9]+)$
 out.$3.$ecs_container_name false
    Emitter_Name        re_emitted

[OUTPUT]
    Name                cloudwatch_logs
    Match               out.*
    region              region
    log_group_name      fallback-group
    log_group_template  /aws/ecs/$ecs_cluster.$ecs_task_definition_family
    log_stream_prefix   task-
    auto_create_group   On
```

Every log which gets into Fluent Bit has a tag which you specify, or is automatically generated when you do not supply one. The tags can be used to route different logs to different destinations. For additional information, see Tag in the *Fluent Bit Official Manual*.

The Fluent Bit configuration described above has the following properties:

- The forward input plug-in listens for incoming traffic on TCP port 24224.
- Each log entry received on that port has a tag which the forward input plug-in modifies to prefix the record with `ecs.` string.
- The Fluent Bit internal pipeline routes the log entry to modify the filter using the Match regex. This filter replaces the keys in the log record JSON to the format which Fluent Bit can consume.

- The modified log entry is then consumed by the rewrite_tag filter. This filter changes the tag of the log record to the format out.*TASK_ID*.*CONTAINER_NAME*.
- The new tag will be routed to output cloudwatch_logs plug-in which creates the log groups and streams as described earlier by using the `log_group_template` and `log_stream_prefix` options of the CloudWatch output plug-in. For additional information, see [Configuration parameters]() in the *Fluent Bit Official Manual*.

Step 4: Register a Windows Fluent Bit task definition which routes the logs to CloudWatch

Register a Windows Fluent Bit task definition which routes the logs to CloudWatch.

> **Note**
>
> This task definition exposes Fluent Bit container port 24224 to the host port 24224. Verify that this port is not open in your EC2 instance security group to prevent access from outside.

To register a task definition

1. Create a file named `fluent-bit.json` with the following contents.

   Replace the following variables:

   - *task-iam-role* with the Amazon Resource Name (ARN) of your task IAM role
   - *region* with the Region where your task runs

```
{
  "family": "ecs-windows-fluent-bit",
  "taskRoleArn": "task-iam-role",
  "containerDefinitions": [
    {
      "name": "fluent-bit",
      "image": "public.ecr.aws/aws-observability/aws-for-fluent-
bit:windowsservercore-latest",
      "cpu": 512,
      "portMappings": [
```

```
        {
          "hostPort": 24224,
          "containerPort": 24224,
          "protocol": "tcp"
        }
      ],
      "entryPoint": [
        "Powershell",
        "-Command"
      ],
      "command": [
        "C:\\entrypoint.ps1 -ConfigFile C:\\ecs_windows_forward_daemon\
\cloudwatch.conf"
      ],
      "environment": [
        {
          "name": "AWS_REGION",
          "value": "region"
        }
      ],
      "memory": 512,
      "essential": true,
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fluent-bit-logs",
          "awslogs-region": "region",
          "awslogs-stream-prefix": "flb",
          "awslogs-create-group": "true"
        }
      }
    }
  ],
  "memory": "512",
  "cpu": "512"
}
```

2. Run the following command to register the task definition.

```
aws ecs register-task-definition --cli-input-json file://fluent-bit.json --
region region
```

You can list the task definitions for your account by running the `list-task-definitions` command. The output of displays the family and revision values that you can use together with `run-task` or `start-task`.

Step 5: Run the `ecs-windows-fluent-bit` task definition as an Amazon ECS service using the daemon scheduling strategy

After you register a task definition for your account, you can run a task in the cluster. For this tutorial, you run one instance of the `ecs-windows-fluent-bit:1` task definition in your `FluentBit-cluster` cluster. Run the task in a service which uses the daemon scheduling strategy, which ensures that a single instance of Fluent Bit always runs on each of your container instances.

To run a task

1. Run the following command to start the `ecs-windows-fluent-bit:1` task definition (registered in the previous step) as a service.

   > **Note**
   >
   > This task definition uses the `awslogs` logging driver, your container instance need to have the necessary permissions.

   Replace the following variables:

   - *`region`* with the Region where your service runs

   ```
   aws ecs create-service \
       --cluster FluentBit-cluster \
       --service-name FluentBitForwardDaemonService \
       --task-definition ecs-windows-fluent-bit:1 \
       --launch-type EC2 \
       --scheduling-strategy DAEMON \
       --region region
   ```

2. Run the following command to list your tasks.

Replace the following variables:

- *region* with the Region where your service tasks run

```
aws ecs list-tasks --cluster FluentBit-cluster --region region
```

Step 6: Register a Windows task definition which generates the logs

Register a task definition which generates the logs. This task definition deploys Windows container image which will write a incremental number to `stdout` every second.

The task definition uses the fluentd logging driver which connects to port 24224 which the Fluent Bit plug-in listens to. The Amazon ECS agent labels each Amazon ECS container with tags including the cluster name, task ARN, task definition family name, task definition revision number and the task container name. These key-value labels are passed to Fluent Bit.

> **Note**
>
> This task uses the `default` network mode. However, you can also use the `awsvpc` network mode with the task.

To register a task definition

1. Create a file named `windows-app-task.json` with the following contents.

```
{
  "family": "windows-app-task",
  "containerDefinitions": [
    {
      "name": "sample-container",
      "image": "mcr.microsoft.com/windows/servercore:ltsc2019",
      "cpu": 512,
      "memory": 512,
      "essential": true,
      "entryPoint": [
        "Powershell",
        "-Command"
```

```
      ],
      "command": [
        "$count=1;while(1) { Write-Host $count; sleep 1; $count=$count+1;}"
      ],
      "logConfiguration": {
        "logDriver": "fluentd",
        "options": {
          "fluentd-address": "localhost:24224",
          "tag": "{{ index .ContainerLabels \"com.amazonaws.ecs.task-definition-
family\" }}",
          "fluentd-async": "true",
          "labels": "com.amazonaws.ecs.cluster,com.amazonaws.ecs.container-
name,com.amazonaws.ecs.task-arn,com.amazonaws.ecs.task-definition-
family,com.amazonaws.ecs.task-definition-version"
        }
      }
    }
  ],
  "memory": "512",
  "cpu": "512"
}
```

2. Run the following command to register the task definition.

   Replace the following variables:

   - *region* with the Region where your task runs

   ```
   aws ecs register-task-definition --cli-input-json file://windows-app-task.json --
   region region
   ```

   You can list the task definitions for your account by running the `list-task-definitions` command. The output of displays the family and revision values that you can use together with `run-task` or `start-task`.

Step 7: Run the `windows-app-task` task definition

After you register the `windows-app-task` task definition, run it in your `FluentBit-cluster` cluster.

To run a task

1. Run the `windows-app-task:1` task definition you registered in the previous step.

   Replace the following variables:

   - *region* with the Region where your task runs

   ```
   aws ecs run-task --cluster FluentBit-cluster --task-definition windows-app-task:1
    --count 2 --region region
   ```

2. Run the following command to list your tasks.

   ```
   aws ecs list-tasks --cluster FluentBit-cluster
   ```

Step 8: Verify the logs on CloudWatch

In order to verify your Fluent Bit setup, check for the following log groups in the CloudWatch console:

- `/ecs/fluent-bit-logs` - This is the log group which corresponds to the Fluent Bit daemon container which is running on the container instance.
- `/aws/ecs/FluentBit-cluster.windows-app-task` - This is the log group which corresponds to all the tasks launched for `windows-app-task` task definition family inside `FluentBit-cluster` cluster.

  `task-out.`*FIRST_TASK_ID*`.sample-container` - This log stream contains all the logs generated by the first instance of the task in the sample-container task container.

  `task-out.`*SECOND_TASK_ID*`.sample-container` - This log stream contains all the logs generated by the second instance of the task in the sample-container task container.

The `task-out.`*TASK_ID*`.sample-container` log stream has fields similar to the following:

```
{
    "source": "stdout",
    "ecs_task_arn": "arn:aws:ecs:region:0123456789012:task/FluentBit-
cluster/13EXAMPLE",
```

```
    "container_name": "/ecs-windows-app-task-1-sample-container-cEXAMPLE",
    "ecs_cluster": "FluentBit-cluster",
    "ecs_container_name": "sample-container",
    "ecs_task_definition_version": "1",
    "container_id": "61f5e6EXAMPLE",
    "log": "10",
    "ecs_task_definition_family": "windows-app-task"
}
```

To verify the Fluent Bit setup

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Log groups**. Make sure that you're in the Region where you deployed Fluent Bit to your containers.

   In the list of log groups in the AWS Region, you should see the following:

   - `/ecs/fluent-bit-logs`
   - `/aws/ecs/FluentBit-cluster.windows-app-task`

   If you see these log groups, the Fluent Bit setup is verified.

Step 9: Clean up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using.

To clean up the tutorial resources

1. Stop the `windows-simple-task` task and the `ecs-fluent-bit` task. For more information, see the section called "Stopping a task".
2. Run the following command to delete the `/ecs/fluent-bit-logs` log group. For more information, about deleting log groups see delete-log-group in the *AWS Command Line Interface Reference*.

   ```
   aws logs delete-log-group --log-group-name /ecs/fluent-bit-logs
   aws logs delete-log-group --log-group-name /aws/ecs/FluentBit-cluster.windows-app-task
   ```

3. Run the following command to terminate the instance.

```
aws ec2 terminate-instances --instance-ids instance-id
```

4. Run the following commands to delete the IAM roles.

```
aws iam delete-role --role-name ecsInstanceRole
aws iam delete-role --role-name fluentTaskRole
```

5. Run the following command to delete the Amazon ECS cluster.

```
aws ecs delete-cluster --cluster FluentBit-cluster
```

Using gMSA for EC2 Linux containers on Amazon ECS

Amazon ECS supports Active Directory authentication for Linux containers on EC2 through a special kind of service account called a *group Managed Service Account* (gMSA).

Linux based network applications, such as .NET Core applications, can use Active Directory to facilitate authentication and authorization management between users and services. You can use this feature by designing applications that integrate with Active Directory and run on domain-joined servers. But, because Linux containers can't be domain-joined, you need to configure a Linux container to run with gMSA.

A Linux container that runs with gMSA relies on the `credentials-fetcher` daemon that runs on the container's host Amazon EC2 instance. That is, the daemon retrieves the gMSA credentials from the Active Directory domain controller and then transfers these credentials to the container instance. For more information about service accounts, see Create gMSAs for Windows containers on the Microsoft Learn website.

Considerations

Consider the following before you use gMSA for Linux containers:

- If your containers run on EC2, you can use gMSA for Windows containers and Linux containers. For information about how to use gMSA for Linux container on Fargate, see Using gMSA for Linux containers on Fargate.
- You might need a Windows computer that's joined to the domain to complete the prerequisites. For example, you might need a Windows computer that's joined to the domain to create the

gMSA in Active Directory with PowerShell. The RSAT Active Director PowerShell tools are only available for Windows. For more information, see [Installing the Active Directory administration tools](#).

- You chose between **domainless gMSA** and **joining each instance to a single domain**. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

  Then, choose the data storage for the CredSpec and optionally, for the Active Directory user credentials for domainless gMSA.

  Amazon ECS uses an Active Directory credential specification file (CredSpec). This file contains the gMSA metadata that's used to propagate the gMSA account context to the container. You generate the CredSpec file and then store it in one of the CredSpec storage options in the following table, specific to the Operating System of the container instances. To use the domainless method, an optional section in the CredSpec file can specify credentials in one of the *domainless user credentials* storage options in the following table, specific to the Operating System of the container instances.

Storage location	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	domainless user credentials	domainless user credentials
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, domainless user credentials
Local file	N/A	CredSpec

Prerequisites

Before you use the gMSA for Linux containers feature with Amazon ECS, make sure to complete the following:

- You set up an Active Directory domain with the resources that you want your containers to access. Amazon ECS supports the following setups:

- An AWS Directory Service Active Directory. AWS Directory Service is an AWS managed Active Directory that's hosted on Amazon EC2. For more information, see Getting Started with AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.
- An on-premises Active Directory. You must ensure that the Amazon ECS Linux container instance can join the domain. For more information, see AWS Direct Connect.

- You have an existing gMSA account in the Active Directory. For more information, see Using gMSA for EC2 Linux containers on Amazon ECS.
- You installed and are running the `credentials-fetcher` daemon on an Amazon ECS Linux container instance. You also added an initial set of credentials to the `credentials-fetcher` daemon to authenticate with the Active Directory.

> **Note**
>
> The `credentials-fetcher` daemon is only available for Amazon Linux 2023 and Fedora 37 and later. The daemon isn't available for Amazon Linux 2. For more information, see aws/credentials-fetcher on GitHub.

- You set up the credentials for the `credentials-fetcher` daemon to authenticate with the Active Directory. The credentials must be a member of the Active Directory security group that has access to the gMSA account. There are multiple options in Decide if you want to join the instances to the domain, or use domainless gMSA..
- You added the required IAM permissions. The permissions that are required depend on the methods that you choose for the initial credentials and for storing the credential specification:
  - If you use *domainless gMSA* for initial credentials, IAM permissions for AWS Secrets Manager are required on the task execution role.
  - If you store the credential specification in SSM Parameter Store, IAM permissions for Amazon EC2 Systems Manager Parameter Store are required on the task execution role.
  - If you store the credential specification in Amazon S3, IAM permissions for Amazon Simple Storage Service are required on the task execution role.

Setting up gMSA-capable Linux Containers on Amazon ECS

Prepare the infrastructure

The following steps are considerations and setup that are performed once. After you complete these steps, you can automate creating container instances to reuse this configuration.

Decide how the initial credentials are provided and configure the EC2 user data in a reusable EC2 launch template to install the `credentials-fetcher` daemon.

1. **Decide if you want to join the instances to the domain, or use domainless gMSA.**

   - **Join EC2 instances to the Active Directory domain**

     - **Join the instances by user data**

       Add the steps to join the Active Directory domain to your EC2 user data in an EC2 launch template. Multiple Amazon EC2 Auto Scaling groups can use the same launch template.

       You can use these steps Joining an Active Directory or FreeIPA domain in the Fedora Docs.

   - **Make an Active Directory user for domainless gMSA**

     The `credentials-fetcher` daemon has a feature that's called *domainless gMSA*. This feature requires a domain, but the EC2 instance doesn't need to be joined to the domain. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance. Instead, you provide the name of a secret in AWS Secrets Manager in the CredSpec file. The secret must contain a username, password, and the domain to log in to.

     This feature is supported and can be used with Linux and Windows containers.

     This feature is similar to the *gMSA support for non-domain-joined container hosts* feature. For more information about the Windows feature, see gMSA architecture and improvements on the Microsoft Learn website.

     a. Make a user in your Active Directory domain. The user in Active Directory must have permission to access the gMSA service accounts that you use in the tasks.

     b. Create a secret in AWS Secrets Manager, after you made the user in Active Directory. For more information, see Create an AWS Secrets Manager secret.

     c. Enter the user's username, password, and the domain into JSON key-value pairs called `username`, `password` and `domainName`, respectively.

     ```
     {"username":"username","password":"passw0rd", "domainName":"example.com"}
     ```

d. Add configuration to the CredSpec file for the service account. The additional `HostAccountConfig` contains the Amazon Resource Name (ARN) of the secret in Secrets Manager.

   On Windows, the `PluginGUID` must match the GUID in the following example snippet. On Linux, the `PluginGUID` is ignored. Replace `MySecret` with example with the Amazon Resource Name (ARN) of your secret.

```
    "ActiveDirectoryConfig": {
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
```

e. The *domainless gMSA* feature needs additional permissions in the task execution role. Follow the step [(Optional) domainless gMSA secret].

2. **Configure instances and install `credentials-fetcher` daemon**

   You can install the `credentials-fetcher` daemon with a user data script in your EC2 Launch Template. The following examples demonstrate two types of user data, `cloud-config` YAML or bash script. These examples are for Amazon Linux 2023 (AL2023). Replace `MyCluster` with the name of the Amazon ECS cluster that you want these instances to join.

   - **`cloud-config` YAML**

```
Content-Type: text/cloud-config
package_reboot_if_required: true
packages:
  # prerequisites
  - dotnet
  - realmd
  - oddjob
  - oddjob-mkhomedir
  - sssd
  - adcli
  - krb5-workstation
  - samba-common-tools
```

```
  # https://github.com/aws/credentials-fetcher gMSA credentials management for
 containers
  - credentials-fetcher
write_files:
# configure the ECS Agent to join your cluster.
# replace MyCluster with the name of your cluster.
- path: /etc/ecs/ecs.config
  owner: root:root
  permissions: '0644'
  content: |
    ECS_CLUSTER=MyCluster
    ECS_GMSA_SUPPORTED=true
runcmd:
# start the credentials-fetcher daemon and if it succeeded, make it start after
 every reboot
- "systemctl start credentials-fetcher"
- "systemctl is-active credentials-fetcher && systemctl enable credentials-
fetcher"
```

- **bash script**

  If you're more comfortable with bash scripts and have multiple variables to write to `/etc/ecs/ecs.config`, use the following `heredoc` format. This format writes everything between the lines beginning with **cat** and `EOF` to the configuration file.

```
#!/usr/bin/env bash
set -euxo pipefail

# prerequisites
timeout 30 dnf install -y dotnet realmd oddjob oddjob-mkhomedir sssd adcli
 krb5-workstation samba-common-tools
# install https://github.com/aws/credentials-fetcher gMSA credentials
 management for containers
timeout 30 dnf install -y credentials-fetcher

# start credentials-fetcher
systemctl start credentials-fetcher
systemctl is-active credentials-fetcher && systemctl enable credentials-fetcher

cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_GMSA_SUPPORTED=true
```

```
EOF
```

There are optional configuration variables for the `credentials-fetcher` daemon that you can set in `/etc/ecs/ecs.config`. We recommend that you set the variables in the user data in the YAML block or `heredoc` similar to the previous examples. Doing so prevents issues with partial configuration that can happen with editing a file multiple times. For more information about the ECS agent configuration, see [Amazon ECS Container Agent](#) on GitHub.

- Optionally, you can use the variable `CREDENTIALS_FETCHER_HOST` if you change the `credentials-fetcher` daemon configuration to move the socket to another location.

Setting up permissions and secrets

Do the following steps once for each application and each task definition. We recommend that you use the best practice of granting the least privilege and narrow the permissions used in the policy. This way, each task can only read the secrets that it needs.

1. **(Optional) domainless gMSA secret**

   If you use the domainless method where the instance isn't joined to the domain, follow this step.

   You must add the following permissions as an inline policy to the task execution IAM role. Doing so gives the `credentials-fetcher` daemon access to the Secrets Manager secret. Replace the `MySecret` example with the Amazon Resource Name (ARN) of your secret in the `Resource` list.

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:my-secret-AbCdEf"
```

```
        }
    ]
}
```

> **Note**
>
> If you use your own KMS key to encrypt your secret, you must add the necessary permissions to this role and add this role to the AWS KMS key policy.

2. **Decide if you're using SSM Parameter Store or S3 to store the CredSpec**

   Amazon ECS supports the following ways to reference the file path in the `credentialSpecs` field of the task definition.

   If you join the instances to a single domain, use the prefix `credentialspec:` at the start of the ARN in the string. If you use domainless gMSA, then use `credentialspecdomainless:`.

   For more information about the CredSpec, see Credential specification file.

   - **Amazon S3 Bucket**

     Add the credential spec to an Amazon S3 bucket. Then, reference the Amazon Resource Name (ARN) of the Amazon S3 bucket in the `credentialSpecs` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/
${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

To give your tasks access to the S3 bucket, add the following permissions as an inline policy to the Amazon ECS task execution IAM role.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/{object}"
            ]
        }
    ]
}
```

- **SSM Parameter Store parameter**

  Add the credential spec to an SSM Parameter Store parameter. Then, reference the Amazon Resource Name (ARN) of the SSM Parameter Store parameter in the `credentialSpecs` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:ssm:aws-region:111122223333:parameter/parameter_name"
            ],
            ...
```

```
        }
    ],
    ...
}
```

To give your tasks access to the SSM Parameter Store parameter, add the following permissions as an inline policy to the Amazon ECS task execution IAM role.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": "arn:aws:ssm:us-east-1:123456789012:parameter/my-
parameter"
        }
    ]
}
```

Credential specification file

Amazon ECS uses an Active Directory credential specification file (*CredSpec*). This file contains the gMSA metadata that's used to propagate the gMSA account context to the Linux container. You generate the CredSpec and reference it in the `credentialSpecs` field in your task definition. The CredSpec file doesn't contain any secrets.

The following is an example CredSpec file.

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
```

```
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "example.com"
            }
        ],
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
    }
}
```

Creating a CredSpec

You create a CredSpec by using the CredSpec PowerShell module on a Windows computer that's joined to the domain. Follow the steps in Create a credential spec on the Microsoft Learn website.

Using gMSA for Linux containers on Fargate

Amazon ECS supports Active Directory authentication for Linux containers on Fargate through a special kind of service account called a *group Managed Service Account* (gMSA).

Linux based network applications, such as .NET Core applications, can use Active Directory to facilitate authentication and authorization management between users and services. You can use this feature by designing applications that integrate with Active Directory and run on domain-joined servers. But, because Linux containers can't be domain-joined, you need to configure a Linux container to run with gMSA.

Considerations

Consider the following before you use gMSA for Linux containers on Fargate:

- You must be running Platform Version 1.4 or later.
- You might need a Windows computer that's joined to the domain to complete the prerequisites. For example, you might need a Windows computer that's joined to the domain to create the gMSA in Active Directory with PowerShell. The RSAT Active Director PowerShell tools are only available for Windows. For more information, see Installing the Active Directory administration tools.
- You must use **domainless gMSA**.

  Amazon ECS uses an Active Directory credential specification file (CredSpec). This file contains the gMSA metadata that's used to propagate the gMSA account context to the container. You generate the CredSpec file, and then store it in an Amazon S3 bucket.
- A task can only support one Active Directory.

Prerequisites

Before you use the gMSA for Linux containers feature with Amazon ECS, make sure to complete the following:

- You set up an Active Directory domain with the resources that you want your containers to access. Amazon ECS supports the following setups:
  - An AWS Directory Service Active Directory. AWS Directory Service is an AWS managed Active Directory that's hosted on Amazon EC2. For more information, see Getting Started with AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.
  - An on-premises Active Directory. You must ensure that the Amazon ECS Linux container instance can join the domain. For more information, see AWS Direct Connect.
- You have an existing gMSA account in the Active Directory and a user that has permission to access the gMSA service account. For more information, see Make an Active Directory user for domainless gMSA.
- You have an Amazon S3 bucket. For more information, see Creating a bucket in the *Amazon S3 User Guide*.

Setting up gMSA-capable Linux Containers on Amazon ECS

Prepare the infrastructure

The following steps are considerations and setup that are performed once.

- **Make an Active Directory user for domainless gMSA**

  When you use domainless gMSA, the container isn't joined to the domain. Other applications that run on the container can't use the credentials to access the domain. Tasks that use a different domain can run on the same container. You provide the name of a secret in AWS Secrets Manager in the CredSpec file. The secret must contain a username, password, and the domain to log in to.

  This feature is similar to the *gMSA support for non-domain-joined container hosts* feature. For more information about the Windows feature, see gMSA architecture and improvements on the Microsoft Learn website.

  a. Configure a user in your Active Directory domain. The user in the Active Directory must have permission to access the gMSA service account that you use in the tasks.

  b. You have a VPC and subnets that can resolve the Active Directory domain name. Configure the VPC with DHCP options with the domain name that points to the Active Directory service name. For information about how to configure DHCP options for a VPC, see Work with DHCP option sets in the *Amazon Virtual Private Cloud User Guide*.

  c. Create a secret in AWS Secrets Manager.

  d. Create the credential specification file.

Setting up permissions and secrets

Do the following steps one time for each application and each task definition. We recommend that you use the best practice of granting the least privilege and narrow the permissions used in the policy. This way, each task can only read the secrets that it needs.

1. Make a user in your Active Directory domain. The user in Active Directory must have permission to access the gMSA service accounts that you use in the tasks.
2. After you make the Active Directory user, create a secret in AWS Secrets Manager. For more information, see Create an AWS Secrets Manager secret.
3. Enter the user's username, password, and the domain into JSON key-value pairs called `username`, `password` and `domainName`, respectively.

   ```
   {"username":"username","password":"passw0rd", "domainName":"example.com"}
   ```

4. You must add the following permissions as an inline policy to the task execution IAM role. Doing so gives the `credentials-fetcher` daemon access to the Secrets Manager secret. Replace the `MySecret` example with the Amazon Resource Name (ARN) of your secret in the `Resource` list.

   JSON

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "secretsmanager:GetSecretValue"
               ],
               "Resource": [
               "arn:aws:secretsmanager:us-east-1:111122223333:secret:MySecret"
               ]
           }
       ]
   }
   ```

   > **Note**
   >
   > If you use your own KMS key to encrypt your secret, you must add the necessary permissions to this role and add this role to the AWS KMS key policy.

5. Add the credential spec to an Amazon S3 bucket. Then, reference the Amazon Resource Name (ARN) of the Amazon S3 bucket in the `credentialSpecs` field of the task definition.

   ```
   {
       "family": "",
       "executionRoleArn": "",
       "containerDefinitions": [
           {
               "name": "",
               ...
               "credentialSpecs": [
                   "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
   ```

```
            ],
            ...
        }
    ],
    ...
}
```

To give your tasks access to the S3 bucket, add the following permissions as an inline policy to the Amazon ECS task execution IAM role.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
            "s3:GetObjectVersion",
            "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::{bucket_name}",
                "arn:aws:s3:::{bucket_name}/{object}"
            ]
        }
    ]
}
```

Credential specification file

Amazon ECS uses an Active Directory credential specification file (*CredSpec*). This file contains the gMSA metadata that's used to propagate the gMSA account context to the Linux container. You generate the CredSpec and reference it in the `credentialSpecs` field in your task definition. The CredSpec file doesn't contain any secrets.

The following is an example CredSpec file.

```
{
```

```
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "example.com"
            }
        ],
        "HostAccountConfig": {
            "PortableCcgVersion": "1",
            "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
            "PluginInput": {
                "CredentialArn": "arn:aws:secretsmanager:aws-
region:111122223333:secret:MySecret"
            }
        }
    }
}
```

Creating a CredSpec and uploading it to an Amazon S3

You create a CredSpec by using the CredSpec PowerShell module on a Windows computer that's joined to the domain. Follow the steps in Create a credential spec on the Microsoft Learn website.

After you create the credential specification file, upload it to an Amazon S3 bucket. Copy the CredSpec file to the computer or environment that you are running AWS CLI commands in.

Run the following AWS CLI command to upload the CredSpec to Amazon S3. Replace *amzn-s3-demo-bucket* with the name of your Amazon S3 bucket. You can store the file as an object in any bucket and location, but you must allow access to that bucket and location in the policy that you attach to the task execution role.

For PowerShell, use the following command:

```
$ Write-S3Object -BucketName "amzn-s3-demo-bucket" -Key "ecs-domainless-gmsa-credspec"
 -File "gmsa-cred-spec.json"
```

The following AWS CLI command uses backslash continuation characters that are used by sh and compatible shells.

```
$ aws s3 cp gmsa-cred-spec.json \
s3://amzn-s3-demo-bucket/ecs-domainless-gmsa-credspec
```

Using Amazon ECS Windows containers with domainless gMSA using the AWS CLI

The following tutorial shows how to create an Amazon ECS task that runs a Windows container that has credentials to access Active Directory with the AWS CLI. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

Topics

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see Installing the AWS Command Line Interface.

> **Note**
>
> You can use dual-stack service endpoints to interact with Amazon ECS from the AWS CLI, SDKs, and the Amazon ECS API over both IPv4 and IPv6. For more information, see Using Amazon ECS dual-stack endpoints.

- You set up an Active Directory domain with the resources that you want your containers to access. Amazon ECS supports the following setups:
  - An AWS Directory Service Active Directory. AWS Directory Service is an AWS managed Active Directory that's hosted on Amazon EC2. For more information, see Getting Started with AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.
  - An on-premises Active Directory. You must ensure that the Amazon ECS Linux container instance can join the domain. For more information, see AWS Direct Connect.
- You have a VPC and subnets that can resolve the Active Directory domain name.
- You chose between **domainless gMSA** and **joining each instance to a single domain**. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

  Then, choose the data storage for the CredSpec and optionally, for the Active Directory user credentials for domainless gMSA.

Amazon ECS uses an Active Directory credential specification file (CredSpec). This file contains the gMSA metadata that's used to propagate the gMSA account context to the container. You generate the CredSpec file and then store it in one of the CredSpec storage options in the following table, specific to the Operating System of the container instances. To use the domainless method, an optional section in the CredSpec file can specify credentials in one of the *domainless user credentials* storage options in the following table, specific to the Operating System of the container instances.

Storage location	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	domainless user credentials	domainless user credentials
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, domainless user credentials
Local file	N/A	CredSpec

- (Optional) AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see What is AWS CloudShell? in the *AWS CloudShell User Guide*.

Step 1: Create and configure the gMSA account on Active Directory Domain Services (AD DS)

Create and configure a gMSA account on the Active Directory domain.

> **Note**
>
> This step creates two separate accounts: a Group Managed Service Account (gMSA) that provides the identity for your containers, and a regular user account that is used for domain authentication. These accounts serve different purposes and should have different names.

1. **Generate a Key Distribution Service root key**

> **Note**
>
> If you are using AWS Directory Service, then you can skip this step.

The KDS root key and gMSA permissions are configured with your AWS managed Microsoft AD.

If you have not already created a gMSA Service Account in your domain, you'll need to first generate a Key Distribution Service (KDS) root key. The KDS is responsible for creating, rotating, and releasing the gMSA password to authorized hosts. When the `ccg.exe` needs to retrieve gMSA credentials, it contact KDS to retrieve the current password.

To check if the KDS root key has already been created, run the following PowerShell cmdlet with domain admin privileges on a domain controller using the `ActiveDirectory` PowerShell module. For more information about the module, see ActiveDirectory Module on the Microsoft Learn website.

```
PS C:\> Get-KdsRootKey
```

If the command returns a key ID, you can skip the rest of this step. Otherwise, create the KDS root key by running the following command:

```
PS C:\> Add-KdsRootKey -EffectiveImmediately
```

Although the argument `EffectiveImmediately` to the command implies the key is effective immediately, you need to wait 10 hours before the KDS root key is replicated and available for use on all domain controllers.

2. **Create the gMSA account**

To create the gMSA account and allow the `ccg.exe` to retrieve the gMSA password, run the following PowerShell commands from a Windows Server or client with access to the domain. Replace `ExampleAccount` with the name that you want for your gMSA account, and replace `example-domain` with your Active Directory domain name (for example, if your domain is `contoso.com`, use `contoso`).

a.
```
PS C:\> Install-WindowsFeature RSAT-AD-PowerShell
```

b.
```
PS C:\> New-ADGroup -Name "ExampleAccount Authorized Hosts" -SamAccountName
 "ExampleAccountHosts" -GroupScope DomainLocal
```

c.
```
PS C:\> New-ADServiceAccount -Name "ExampleAccount" -DnsHostName "example-
domain" -ServicePrincipalNames "host/ExampleAccount", "host/example-domain" -
PrincipalsAllowedToRetrieveManagedPassword "ExampleAccountHosts"
```

d. Create a user with a permanent password that doesn't expire. These credentials are stored in AWS Secrets Manager and used by each task to join the domain. This is a separate user account from the gMSA account created above. Replace `ExampleServiceUser` with the name you want for this service user account.

```
PS C:\> New-ADUser -Name "ExampleServiceUser" -AccountPassword (ConvertTo-
SecureString -AsPlainText "Test123" -Force) -Enabled 1 -PasswordNeverExpires 1
```

e.
```
PS C:\> Add-ADGroupMember -Identity "ExampleAccountHosts" -Members
 "ExampleServiceUser"
```

f. Install the PowerShell module for creating CredSpec objects in Active Directory and output the CredSpec JSON.

```
PS C:\> Install-PackageProvider -Name NuGet -Force
```

```
PS C:\> Install-Module CredentialSpec
```

g.
```
PS C:\> New-CredentialSpec -AccountName ExampleAccount
```

3. Copy the JSON output from the previous command into a file called `gmsa-cred-spec.json`. This is the CredSpec file. It is used in Step 3, Step 3: Modify your CredSpec JSON to include domainless gMSA information.

Step 2: Upload Credentials to Secrets Manager

Copy the Active Directory credentials into a secure credential storage system, so that each task retrieves it. This is the domainless gMSA method. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

- Run the following AWS CLI command and replace the username, password, and domain name to match your environment. Use the service user account name (not the gMSA account name) for the username. Keep the ARN of the secret to use in the next step, [Step 3: Modify your CredSpec JSON to include domainless gMSA information](#)

  The following command uses backslash continuation characters that are used by `sh` and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

  ```
  $ aws secretsmanager create-secret \
  --name gmsa-plugin-input \
  --description "Amazon ECS - gMSA Portable Identity." \
  --secret-string "{\"username\":\"ExampleServiceUser\",\"password\":\"Test123\",
  \"domainName\":\"contoso.com\"}"
  ```

Step 3: Modify your CredSpec JSON to include domainless gMSA information

Before uploading the CredSpec to one of the storage options, add information to the CredSpec with the ARN of the secret in Secrets Manager from the previous step. For more information, see [Additional credential spec configuration for non-domain-joined container host use case](#) on the Microsoft Learn website.

1. Add the following information to the CredSpec file inside the `ActiveDirectoryConfig`. Replace the ARN with the secret in Secrets Manager from the previous step.

   Note that the `PluginGUID` value must match the GUID in the following example snippet and is required.

   ```
   "HostAccountConfig": {
         "PortableCcgVersion": "1",
         "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
   ```

```
      "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
```

You can also use a secret in SSM Parameter Store by using the ARN in this format: `\"arn:aws:ssm:aws-region:111122223333:parameter/gmsa-plugin-input\"`.

2. After you modify the CredSpec file, it should look like the following example:

```
{
  "CmsPlugins": [
    "ActiveDirectory"
  ],
  "DomainJoinConfig": {
    "Sid": "S-1-5-21-4066351383-705263209-1606769140",
    "MachineAccountName": "ExampleAccount",
    "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
    "DnsTreeName": "example-domain",
    "DnsName": "example-domain",
    "NetBiosName": "example-domain"
  },
  "ActiveDirectoryConfig": {
    "GroupManagedServiceAccounts": [
      {
        "Name": "ExampleAccount",
        "Scope": "example-domain"
      },
      {
        "Name": "ExampleAccount",
        "Scope": "example-domain"
      }
    ],
    "HostAccountConfig": {
      "PortableCcgVersion": "1",
      "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
      "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
  }
}
```

Step 4: Upload CredSpec to Amazon S3

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

1. Copy the CredSpec file to the computer or environment that you are running AWS CLI commands in.
2. Run the following AWS CLI command to upload the CredSpec to Amazon S3. Replace `MyBucket` with the name of your Amazon S3 bucket. You can store the file as an object in any bucket and location, but you must allow access to that bucket and location in the policy that you attach to the task execution role.

   The following command uses backslash continuation characters that are used by `sh` and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

   ```
   $ aws s3 cp gmsa-cred-spec.json \
   s3://MyBucket/ecs-domainless-gmsa-credspec
   ```

Step 5: (Optional) Create an Amazon ECS cluster

By default, your account has an Amazon ECS cluster named `default`. This cluster is used by default in the AWS CLI, SDKs, and AWS CloudFormation. You can use additional clusters to group and organize tasks and infrastructure, and assign defaults for some configuration.

You can create a cluster from the AWS Management Console, AWS CLI, SDKs, or AWS CloudFormation. The settings and configuration in the cluster don't affect gMSA.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

```
$ aws ecs create-cluster --cluster-name windows-domainless-gmsa-cluster
```

> **Important**
>
> If you choose to create your own cluster, you must specify `--cluster clusterName` for each command that you intend to use with that cluster.

Step 6: Create an IAM role for container instances

A *container instance* is a host computer to run containers in ECS tasks, for example Amazon EC2 instances. Each container instance registers to an Amazon ECS cluster. Before you launch Amazon EC2 instances and register them to a cluster, you must create an IAM role for your container instances to use.

To create the container instance role, see Amazon ECS container instance IAM role. The default `ecsInstanceRole` has sufficient permissions to complete this tutorial.

Step 7: Create a custom task execution role

Amazon ECS can use a different IAM role for the permissions needed to start each task, instead of the container instance role. This role is the *task execution role*. We recommend creating a task execution role with only the permissions required for ECS to run the task, also known as *least-privilege permissions*. For more information about the principle of least privilege, see SEC03-BP02 Grant least privilege access in the *AWS Well-Architected Framework*.

1. To create a task execution role, see Creating the task execution role. The default permissions allow the container instance to pull container images from Amazon Elastic Container Registry and `stdout` and `stderr` from your applications to be logged to Amazon CloudWatch Logs.

   Because the role needs custom permissions for this tutorial, you can give the role a different name than `ecsTaskExecutionRole`. This tutorial uses `ecsTaskExecutionRole` in further steps.
2. Add the following permissions by creating a custom policy, either an inline policy that only exists in for this role, or a policy that you can reuse. Replace the ARN for the `Resource` in the first statement with the Amazon S3 bucket and location, and the second `Resource` with the ARN of the secret in Secrets Manager.

   If you encrypt the secret in Secrets Manager with a custom key, you must also allow `kms:Decrypt` for the key.

   If you use SSM Parameter Store instead of Secrets Manager, you must allow `ssm:GetParameter` for the parameter, instead of `secretsmanager:GetSecretValue`.

   JSON

   ```
   {
   ```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::MyBucket/ecs-domainless-gmsa-credspec/
gmsa-cred-spec.json"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:gmsa-plugin-AbCdEf"
        }
    ]
}
```

Step 8: Create a task role for Amazon ECS Exec

This tutorial uses Amazon ECS Exec to verify functionality by running a command inside a running task. To use ECS Exec, the service or task must turn on ECS Exec and the task role (but not the task execution role) must have `ssmmessages` permissions. For the required IAM policy, see ECS Exec permissions.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

To create a task role using the AWS CLI, follow these steps.

1. Create a file called `ecs-tasks-trust-policy.json` with the following contents:

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Create an IAM role. You can replace the name `ecs-exec-demo-task-role` but keep the name for following steps.

   The following command uses backslash continuation characters that are used by `sh` and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

   ```
   $ aws iam create-role --role-name ecs-exec-demo-task-role \
   --assume-role-policy-document file://ecs-tasks-trust-policy.json
   ```

   You can delete the file `ecs-tasks-trust-policy.json`.

3. Create a file called `ecs-exec-demo-task-role-policy.json` with the following contents:

   JSON

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "ssmmessages:CreateControlChannel",
                   "ssmmessages:CreateDataChannel",
                   "ssmmessages:OpenControlChannel",
                   "ssmmessages:OpenDataChannel"
               ],
               "Resource": "*"
           }
       ]
   }
   ```

4. Create an IAM policy and attach it to the role from the previous step.

The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws iam put-role-policy \
    --role-name ecs-exec-demo-task-role \
    --policy-name ecs-exec-demo-task-role-policy \
    --policy-document file://ecs-exec-demo-task-role-policy.json
```

You can delete the file `ecs-exec-demo-task-role-policy.json`.

Step 9: Register a task definition that uses domainless gMSA

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

1. Create a file called `windows-gmsa-domainless-task-def.json` with the following contents:

```
{
  "family": "windows-gmsa-domainless-task",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 1024,
      "memory": 1024,
      "essential": true,
      "credentialSpecs": [
                 "credentialspecdomainless:arn:aws:s3:::ecs-domainless-gmsa-
credspec/gmsa-cred-spec.json"
      ],
      "entryPoint": [
        "powershell",
        "-Command"
      ],
      "command": [
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -ItemType file -Value
 '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top:
```

```
 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-
align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your
 application is now running on a container in Amazon ECS.</p>' -Force ; C:\
\ServiceMonitor.exe w3svc"
      ],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80,
          "hostPort": 8080
        }
      ]
    }
  ],
  "taskRoleArn": "arn:aws:iam::111122223333:role/ecs-exec-demo-task-role",
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole"
}
```

2. Register the task definition by running the following command:

   The following command uses backslash continuation characters that are used by sh and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws ecs register-task-definition \
--cli-input-json file://windows-gmsa-domainless-task-def.json
```

Step 10: Register a Windows container instance to the cluster

Launch an Amazon EC2 Windows instance and run the ECS container agent to register it as a container instance in the cluster. ECS runs tasks on the container instances that are registered to the cluster that the tasks are started in.

1. To launch an Amazon EC2 Windows instance that is configured for Amazon ECS in the AWS Management Console, see Launching an Amazon ECS Windows container instance. Stop at the step for *user data*.

2. For gMSA, the user data must set the environment variable `ECS_GMSA_SUPPORTED` before starting the ECS container agent.

   For ECS Exec, the agent must start with the argument `-EnableTaskIAMRole`.

To secure the instance IAM role by preventing tasks from reaching the EC2 IMDS web service to retrieve the role credentials, add the argument -AwsvpcBlockIMDS. This only applies to tasks that use the awsvpc network mode.

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_GMSA_SUPPORTED", $TRUE, "Machine")
Import-Module ECSTools
Initialize-ECSAgent -Cluster windows-domainless-gmsa-cluster -EnableTaskIAMRole -
AwsvpcBlockIMDS
</powershell>
```

3. Review a summary of your instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

Step 11: Verify the container instance

You can verify that there is a container instance in the cluster using the AWS Management Console. However, gMSA needs additional features that are indicated as *attributes*. These attributes aren't visible in the AWS Management Console, so this tutorial uses the AWS CLI.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is bash.

1. List the container instances in the cluster. Container instances have an ID that is different from the ID of the EC2 instance.

```
$ aws ecs list-container-instances
```

Output:

```
{
    "containerInstanceArns": [
        "arn:aws:ecs:aws-region:111122223333:container-
instance/default/MyContainerInstanceID"
    ]
}
```

For example, `526bd5d0ced448a788768334e79010fd` is a valid container instance ID.

2. Use the container instance ID from the previous step to get the details for the container instance. Replace `MyContainerInstanceID` with the ID.

   The following command uses backslash continuation characters that are used by `sh` and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

   ```
   $ aws ecs describe-container-instances \
        ----container-instances MyContainerInstanceID
   ```

   Note that the output is very long.

3. Verify that the `attributes` list has an object with the key called `name` and a value `ecs.capability.gmsa-domainless`. The following is an example of the object.

   Output:

   ```
   {
       "name": "ecs.capability.gmsa-domainless"
   }
   ```

Step 12: Run a Windows task

Run an Amazon ECS task. If there is only 1 container instance in the cluster, you can use `run-task`. If there are many different container instances, it might be easier to use `start-task` and specify the container instance ID to run the task on, than to add placement constraints to the task definition to control what type of container instance to run this task on.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

1.
   The following command uses backslash continuation characters that are used by `sh` and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

   ```
   aws ecs run-task --task-definition windows-gmsa-domainless-task \
   ```

```
    --enable-execute-command --cluster windows-domainless-gmsa-cluster
```

Note the task ID that is returned by the command.

2. Run the following command to verify that the task has started. This command waits and doesn't return the shell prompt until the task starts. Replace `MyTaskID` with the task ID from the previous step.

```
$ aws ecs wait tasks-running --task MyTaskID
```

Step 13: Verify the container has gMSA credentials

Verify that the container in the task has a Kerberos token. gMSA

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

1. The following command uses backslash continuation characters that are used by `sh` and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

```
$ aws ecs execute-command \
--task MyTaskID \
--container windows_sample_app \
--interactive \
--command powershell.exe
```

The output will be a PowerShell prompt.

2. Run the following command in the PowerShell terminal inside the container.

```
PS C:\> klist get ExampleAccount$
```

In the output, note the `Principal` is the one that you created previously.

Step 14: Clean up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

This step uses the AWS CLI. You can run these commands in AWS CloudShell in the default shell, which is `bash`.

1. Stop the task. Replace `MyTaskID` with the task ID from step 12, [Step 12: Run a Windows task](#).

   ```
   $ aws ecs stop-task --task MyTaskID
   ```

2. Terminate the Amazon EC2 instance. Afterwards, the container instance in the cluster will be deleted automatically after one hour.

   You can find and terminate the instance by using the Amazon EC2 console. Or, you can run the following command. To run the command, find the EC2 instance ID in the output of the `aws ecs describe-container-instances` command from step 1, [Step 11: Verify the container instance](#). i-10a64379 is an example of an EC2 instance ID.

   ```
   $ aws ec2 terminate-instances --instance-ids MyInstanceID
   ```

3. Delete the CredSpec file in Amazon S3. Replace `MyBucket` with the name of your Amazon S3 bucket.

   ```
   $ aws s3api delete-object --bucket MyBucket --key ecs-domainless-gmsa-credspec/
   gmsa-cred-spec.json
   ```

4. Delete the secret from Secrets Manager. If you used SSM Parameter Store instead, delete the parameter.

   The following command uses backslash continuation characters that are used by `sh` and compatible shells. This command isn't compatible with PowerShell. You must modify the command to use it with PowerShell.

   ```
   $ aws secretsmanager delete-secret --secret-id gmsa-plugin-input \
        --force-delete-without-recovery
   ```

5. Deregister and delete the task definition. By deregistering the task definition, you mark it as inactive so it can't be used to start new tasks. Then, you can delete the task definition.

a. Deregister the task definition by specifying the version. ECS automatically makes versions of task definitions, that are numbered starting from 1. You refer to the versions in the same format as the labels on container images, such as `:1`.

```
$ aws ecs deregister-task-definition --task-definition windows-gmsa-domainless-task:1
```

b. Delete the task definition.

```
$ aws ecs delete-task-definitions --task-definition windows-gmsa-domainless-task:1
```

6. (Optional) Delete the ECS cluster, if you created a cluster.

```
$ aws ecs delete-cluster --cluster windows-domainless-gmsa-cluster
```

Debugging Amazon ECS domainless gMSA for Windows containers

Amazon ECS task status

ECS tries to start a task exactly once. Any task that has an issue is stopped, and set to the status `STOPPED`. There are two common types of issues with tasks. First, tasks that couldn't be started. Second, tasks where the application has stopped inside one of the containers. In the AWS Management Console, look at the **Stopped reason** field of the task for the reason why the task was stopped. In the AWS CLI, describe the task and look at the `stoppedReason`. For steps in the AWS Management Console and AWS CLI, see Viewing Amazon ECS stopped task errors.

Windows Events

Windows Events for gMSA in containers are logged in the `Microsoft-Windows-Containers-CCG` log file and can be found in the Event Viewer in the section Applications and Services in `Logs\Microsoft\Windows\Containers-CCG\Admin`. For more debugging tips, see Troubleshoot gMSAs for Windows containers on the Microsoft Learn website.

ECS agent gMSA plugin

Logging for gMSA plugin for the ECS agent on the Windows container instance is in the following directory, `C:/ProgramData/Amazon/gmsa-plugin/`. Look in this log to see if the domainless user credentials were downloaded from the storage location, such as Secrets Manager, and that the credential format was read correctly.

Learn how to use gMSAs for EC2 Windows containers for Amazon ECS

Amazon ECS supports Active Directory authentication for Windows containers through a special kind of service account called a *group Managed Service Account* (gMSA).

Windows based network applications such as .NET applications often use Active Directory to facilitate authentication and authorization management between users and services. Developers commonly design their applications to integrate with Active Directory and run on domain-joined servers for this purpose. Because Windows containers cannot be domain-joined, you must configure a Windows container to run with gMSA.

A Windows container running with gMSA relies on its host Amazon EC2 instance to retrieve the gMSA credentials from the Active Directory domain controller and provide them to the container instance. For more information, see Create gMSAs for Windows containers.

> **Note**
>
> This feature is not supported on Windows containers on Fargate.

Topics

- Considerations
- Prerequisites
- Setting up gMSA for Windows Containers on Amazon ECS

Considerations

The following should be considered when using gMSAs for Windows containers:

- When using the Amazon ECS-optimized Windows Server 2016 Full AMI for your container instances, the container hostname must be the same as the gMSA account name defined in the credential spec file. To specify a hostname for a container, use the `hostname` container definition parameter. For more information, see Network settings.

- You chose between **domainless gMSA** and **joining each instance to a single domain**. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

  Then, choose the data storage for the CredSpec and optionally, for the Active Directory user credentials for domainless gMSA.

  Amazon ECS uses an Active Directory credential specification file (CredSpec). This file contains the gMSA metadata that's used to propagate the gMSA account context to the container. You generate the CredSpec file and then store it in one of the CredSpec storage options in the following table, specific to the Operating System of the container instances. To use the domainless method, an optional section in the CredSpec file can specify credentials in one of the *domainless user credentials* storage options in the following table, specific to the Operating System of the container instances.

Storage location	Linux	Windows
Amazon Simple Storage Service	CredSpec	CredSpec
AWS Secrets Manager	domainless user credentials	domainless user credentials
Amazon EC2 Systems Manager Parameter Store	CredSpec	CredSpec, domainless user credentials
Local file	N/A	CredSpec

Prerequisites

Before you use the gMSA for Windows containers feature with Amazon ECS, make sure to complete the following:

- You set up an Active Directory domain with the resources that you want your containers to access. Amazon ECS supports the following setups:
  - An AWS Directory Service Active Directory. AWS Directory Service is an AWS managed Active Directory that's hosted on Amazon EC2. For more information, see [Getting Started with AWS Managed Microsoft AD](#) in the *AWS Directory Service Administration Guide*.

  - An on-premises Active Directory. You must ensure that the Amazon ECS Linux container instance can join the domain. For more information, see AWS Direct Connect.
- You have an existing gMSA account in the Active Directory. For more information, see Create gMSAs for Windows containers.
- **You chose to use *domainless gMSA* or the Amazon ECS Windows container instance hosting the Amazon ECS task must be *domain joined* to the Active Directory and be a member of the Active Directory security group that has access to the gMSA account.**

  By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.
- You added the required IAM permissions. The permissions that are required depend on the methods that you choose for the initial credentials and for storing the credential specification:
  - If you use *domainless gMSA* for initial credentials, IAM permissions for AWS Secrets Manager are required on the Amazon EC2 instance role.
  - If you store the credential specification in SSM Parameter Store, IAM permissions for Amazon EC2 Systems Manager Parameter Store are required on the task execution role.
  - If you store the credential specification in Amazon S3, IAM permissions for Amazon Simple Storage Service are required on the task execution role.

Setting up gMSA for Windows Containers on Amazon ECS

To set up gMSA for Windows Containers on Amazon ECS, you can follow the complete tutorial that includes configuring the prerequisites Using Amazon ECS Windows containers with domainless gMSA using the AWS CLI.

The following sections cover the CredSpec configuration in detail.

Topics

Example CredSpec

Amazon ECS uses a credential spec file that contains the gMSA metadata used to propagate the gMSA account context to the Windows container. You can generate the credential spec file and reference it in the `credentialSpec` field in your task definition. The credential spec file does not contain any secrets.

The following is an example credential spec file:

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "contoso.com",
        "DnsName": "contoso.com",
        "NetBiosName": "contoso"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "contoso.com"
            }
        ]
    }
}
```

Domainless gMSA setup

We recommend domainless gMSA instead of joining the container instances to a single domain. By using domainless gMSA, the container instance isn't joined to the domain, other applications on the instance can't use the credentials to access the domain, and tasks that join different domains can run on the same instance.

1. Before uploading the CredSpec to one of the storage options, add information to the CredSpec with the ARN of the secret in Secrets Manager or SSM Parameter Store. For more information,

see [Additional credential spec configuration for non-domain-joined container host use case]() on the Microsoft Learn website.

Domainless gMSA credential format

The following is the JSON format for the domainless gMSA credentials for your Active Directory. Store the credentials in Secrets Manager or SSM Parameter Store.

```
{
    "username":"WebApp01",
    "password":"Test123!",
    "domainName":"contoso.com"
}
```

2. Add the following information to the CredSpec file inside the `ActiveDirectoryConfig`. Replace the ARN with the secret in Secrets Manager or SSM Parameter Store.

   Note that the `PluginGUID` value must match the GUID in the following example snippet and is required.

```

    "HostAccountConfig": {
        "PortableCcgVersion": "1",
        "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
        "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
    }
```

   You can also use a secret in SSM Parameter Store by using the ARN in this format: `\"arn:aws:ssm:aws-region:111122223333:parameter/gmsa-plugin-input\"`.

3. After you modify the CredSpec file, it should look like the following example:

```
{
  "CmsPlugins": [
    "ActiveDirectory"
  ],
  "DomainJoinConfig": {
    "Sid": "S-1-5-21-4066351383-705263209-1606769140",
    "MachineAccountName": "WebApp01",
    "Guid": "ac822f13-583e-49f7-aa7b-284f9a8c97b6",
    "DnsTreeName": "contoso",
```

```
      "DnsName": "contoso",
      "NetBiosName": "contoso"
    },
    "ActiveDirectoryConfig": {
      "GroupManagedServiceAccounts": [
        {
          "Name": "WebApp01",
          "Scope": "contoso"
        },
        {
          "Name": "WebApp01",
          "Scope": "contoso"
        }
      ],
      "HostAccountConfig": {
        "PortableCcgVersion": "1",
        "PluginGUID": "{859E1386-BDB4-49E8-85C7-3070B13920E1}",
        "PluginInput": "{\"credentialArn\": \"arn:aws:secretsmanager:aws-
region:111122223333:secret:gmsa-plugin-input\"}"
      }
    }
}
```

Referencing a Credential Spec File in a Task Definition

Amazon ECS supports the following ways to reference the file path in the `credentialSpecs` field of the task definition. For each of these options, you can provide `credentialspec:` or `domainlesscredentialspec:`, depending on whether you are joining the container instances to a single domain, or using domainless gMSA, respectively.

Amazon S3 Bucket

Add the credential spec to an Amazon S3 bucket and then reference the Amazon Resource Name (ARN) of the Amazon S3 bucket in the `credentialSpecs` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
```

```
            "credentialSpecs": [
                "credentialspecdomainless:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the Amazon S3 bucket.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{bucket_name}",
                "arn:aws:s3:::{bucket_name}/{object}"
            ]
        }
    ]
}
```

SSM Parameter Store parameter

Add the credential spec to an SSM Parameter Store parameter and then reference the Amazon Resource Name (ARN) of the SSM Parameter Store parameter in the `credentialSpecs` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
```

```
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
 "credentialspecdomainless:arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ],
            ...
        }
    ],
    ...
}
```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the SSM Parameter Store parameter.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": "arn:aws:ssm:*:111122223333:parameter/example*"
        }
    ]
}
```

Local File

With the credential spec details in a local file, reference the file path in the `credentialSpecs` field of the task definition. The file path referenced must be relative to the `C:\ProgramData\Docker\CredentialSpecs` directory and use the backslash ('\') as the file path separator.

```
{
    "family": "",
    "executionRoleArn": "",
```

```
    "containerDefinitions": [
        {
            "name": "",
            ...
            "credentialSpecs": [
                "credentialspec:file://CredentialSpecDir\CredentialSpecFile.json"
            ],
            ...
        }
    ],
    ...
}
```

Using EC2 Image Builder to build customized Amazon ECS-optimized AMIs

AWS recommends that you use the Amazon ECS-optimized AMIs because they are preconfigured with the requirements and recommendations to run your container workloads. There might be times when you need to customize your AMI to add additional software. You can use EC2 Image Builder for the creation, management, and deployment of your server images. You retain ownership of the customized images created in your account. You can use EC2 Image Builder pipelines to automate updates and system patching for your images, or use a stand-alone command to create an image with your defined configuration resources.

You create a recipe for your image. The recipe includes a parent image, and any additional components. You also create a pipeline which distributes your customized AMI.

You create a recipe for your image. An Image Builder image recipe is a document that defines the base image and the components that are applied to the base image to produce the desired configuration for the output AMI image. You also create a pipeline which distributes your customized AMI. For more information, see How EC2 Image Builder works in the *EC2 Image Builder User Guide*.

We recommend that you use one of the following Amazon ECS-optimized AMIs as your "Parent image" in EC2 Image Builder:

- Linux
  - Amazon ECS-optimized AL2023 x86
  - Amazon ECS-optimized Amazon Linux 2023 (arm64) AMI

  - Amazon ECS-optimized Amazon Linux 2 kernel 5 AMI
  - Amazon ECS-optimized Amazon Linux 2 x86 AMI
- Windows
  - Amazon ECS-optimized Windows 2022 Full x86
  - Amazon ECS-optimized Windows 2022 Core x86
  - Amazon ECS-optimized Windows 2019 Full x86
  - Amazon ECS-optimized Windows 2019 Core x86
  - Amazon ECS-optimized Windows 2016 Full x86

We also recommend that you select "Use latest available OS version". The pipeline will use semantic versioning for the parent image, which helps detect the dependency updates in automatically scheduled jobs. For more information, see Semantic versioning in the *EC2 Image Builder User Guide*.

AWS regularly updates Amazon ECS-optimized AMI images with security patches and the new container agent version. When you use an AMI ID as your parent image in your image recipe, you need to regularly check for updates to the parent image. If there are updates, you must create a new version of your recipe with the updated AMI. This ensures that your custom images incorporate the latest version of the parent image. For information about how to create a workflow to automatically update your EC2 instances in your Amazon ECS cluster with the newly created AMIs, see How to create an AMI hardening pipeline and automate updates to your ECS instance fleet.

You can also specify the Amazon Resource Name (ARN) of a parent image that's published through a managed EC2 Image Builder pipeline. Amazon routinely publishes Amazon ECS-optimized AMI images through managed pipelines. These images are publicly accessible. You must have the correct permissions to access the image. When you use an image ARN instead of an AMI in your Image Builder recipe, your pipeline automatically uses the most recent version of the parent image each time it runs. This approach eliminates the need to manually create new recipe versions for each update.

Using the image ARN with infrastructure as code (IaC)

You can configure the recipe using the EC2 Image Builder console, or infrastructure as code (for example, AWS CloudFormation), or the AWS SDK. When you specify a parent image in your recipe, you can specify an EC2 AMI ID, Image Builder image ARN, AWS Marketplace product ID, or container image. AWS publishes both AMI IDs and Image Builder image ARNs of Amazon ECS-Optimized AMIs publicly. The following is the ARN format for the image:

```
arn:${Partition}:imagebuilder:${Region}:${Account}:image/${ImageName}/${ImageVersion}
```

The `ImageVersion` has the following format. Replace *`major`*, *`minor`* and *`patch`* with the latest values.

```
<major>.<minor>.<patch>
```

You can replace `major`, `minor` and `patch` to specific values or you can use Versionless ARN of an image, so your pipeline remains up-to-date with the latest version of the parent image. A versionless ARN uses wildcard format 'x.x.x' to represent the image version. This approach allows the Image Builder service to automatically resolve to the most recent version of the image. Using versionless ARN ensures that your reference always point to the latest image available, streamlining the process of maintaining up to date images in your deployment. When you create a recipe with the console, EC2 Image Builder automatically identifies the ARN for your parent image. When you use IaC to create the recipe, you must identify the ARN and select the desired image version or use versionless arn to indicate latest available image. We recommend that you create an automated script to filter and only display images that align with your criteria. The following Python script shows how to retrieve a list of Amazon ECS-optimized AMIs.

The script accepts two optional arguments: `owner` and `platform`, with default values of "Amazon", and "Windows" respectively. The valid values for the owner argument are: `Self`, `Shared`, `Amazon`, and `ThirdParty`. The valid values for the platform argument are `Windows` and `Linux`. For example, if you run the script with the `owner` argument set to `Amazon` and the `platform` set to `Linux`, the script generates a list of images published by Amazon including Amazon ECS-Optimized images.

```
import boto3
import argparse

def list_images(owner, platform):
    # Create a Boto3 session
    session = boto3.Session()

    # Create an EC2 Image Builder client
    client = session.client('imagebuilder')
```

```
    # Define the initial request parameters
    request_params = {
        'owner': owner,
        'filters': [
            {
                'name': 'platform',
                'values': [platform]
            }
        ]
    }

    # Initialize the results list
    all_images = []

    # Get the initial response with the first page of results
    response = client.list_images(**request_params)

    # Extract images from the response
    all_images.extend(response['imageVersionList'])

    # While 'nextToken' is present, continue paginating
    while 'nextToken' in response and response['nextToken']:
        # Update the token for the next request
        request_params['nextToken'] = response['nextToken']

        # Get the next page of results
        response = client.list_images(**request_params)

        # Extract images from the response
        all_images.extend(response['imageVersionList'])

    return all_images

def main():
    # Initialize the parser
    parser = argparse.ArgumentParser(description="List AWS images based on owner and
 platform")

    # Add the parameters/arguments
    parser.add_argument("--owner", default="Amazon", help="The owner of the images.
 Default is 'Amazon'.")
    parser.add_argument("--platform", default="Windows", help="The platform type of the
 images. Default is 'Windows'.")
```

```
    # Parse the arguments
    args = parser.parse_args()

    # Retrieve all images based on the provided owner and platform
    images = list_images(args.owner, args.platform)

    # Print the details of the images
    for image in images:
        print(f"Name: {image['name']}, Version: {image['version']}, ARN:
 {image['arn']}")

if __name__ == "__main__":
    main()
```

Using the image ARN with AWS CloudFormation

An Image Builder image recipe is a blueprint that specifies the parent image and components required to achieve the intended configuration of the output image. You use the `AWS::ImageBuilder::ImageRecipe` resource. Set the `ParentImage` value to the image ARN. Use the versionless ARN of your desired image to ensure your pipeline always uses the most recent version of the image. For example, `arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2023-ecs-optimized-x86/x.x.x`. The following `AWS::ImageBuilder::ImageRecipe` resource definition uses an Amazon managed image ARN:

```
ECSRecipe:
    Type: AWS::ImageBuilder::ImageRecipe
    Properties:
      Name: MyRecipe
      Version: '1.0.0'
      Components:
        - ComponentArn: [<The component arns of the image recipe>]
      ParentImage: "arn:aws:imagebuilder:us-east-1:aws:image/amazon-linux-2023-ecs-
optimized-x86/x.x.x"
```

For more information about the [`AWS::ImageBuilder::ImageRecipe`] resource see in the *AWS CloudFormation User Guide*.

You can automate the creation of new images in your pipeline by setting the `Schedule` property of the `AWS::ImageBuilder::ImagePipeline` resource. The schedule includes a start condition and cron expression. For more information, see [AWS::ImageBuilder::ImagePipeline] in the *AWS CloudFormation User Guide*.

The following of `AWS::ImageBuilder::ImagePipeline` example has the pipeline run a build at 10:00AM Coordinated Universal Time (UTC) every day. Set the following `Schedule` values:

- Set `PipelineExecutionStartCondition` to `EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE`. The build initiates only if dependent resources like the parent image or components, which use the wildcard ‘x’ in their semantic versions, are updated. This ensures the build incorporates the latest updates of those resources.
- Set ScheduleExpression to the cron expression `(0 10 * * ? *)`.

```
ECSPipeline:
    Type: AWS::ImageBuilder::ImagePipeline
    Properties:
      Name: my-pipeline
      ImageRecipeArn: <arn of the recipe you created in previous step>
      InfrastructureConfigurationArn: <ARN of the infrastructure configuration
 associated with this image pipeline>
      Schedule:
        PipelineExecutionStartCondition:
 EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE
        ScheduleExpression: 'cron(0 10 * * ? *)'
```

Using the image ARN with Terraform

The approach for specifying your pipeline's parent image and schedule in Terraform aligns with that in AWS CloudFormation. You use the `aws_imagebuilder_image_recipe` resource. Set the `parent_image` value to the image ARN. Use the versionless ARN of your desired image to ensure your pipeline always uses the most recent version of the image. For more information, see [aws_imagebuilder_image_recipe]in the Terraform documentation.

In the schedule configuration block of the `aws_imagebuilder_image_pipeline resource`, set the `schedule_expression` argument value to a cron expression of your choice to specify how often the pipeline runs, and set the `pipeline_execution_start_condition` to `EXPRESSION_MATCH_AND_DEPENDENCY_UPDATES_AVAILABLE`. For more information, see [aws_imagebuilder_image_pipeline]in the Terraform documentation.

Using AWS Deep Learning Containers on Amazon ECS

AWS Deep Learning Containers provide a set of Docker images for training and serving models in TensorFlow and Apache MXNet (Incubating) on Amazon ECS. Deep Learning Containers enable optimized environments with TensorFlow, NVIDIA CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries. Container images for Deep Learning Containers are available in Amazon ECR to reference in Amazon ECS task definitions. You can use Deep Learning Containers along with Amazon Elastic Inference to lower your inference costs.

To get started using Deep Learning Containers without Elastic Inference on Amazon ECS, see Amazon ECS setup in the *AWS Deep Learning AMIs Developer Guide*.

Tutorial: Creating a Service Using a Blue/Green Deployment

Amazon ECS has integrated blue/green deployments into the Create Service wizard on the Amazon ECS console. For more information, see Creating an Amazon ECS rolling update deployment.

The following tutorial shows how to create an Amazon ECS service containing a Fargate task that uses the blue/green deployment type with the AWS CLI.

> **Note**
>
> Support for performing a blue/green deployment has been added for AWS CloudFormation. For more information, see Perform Amazon ECS blue/green deployments through CodeDeploy using AWS CloudFormation in the *AWS CloudFormation User Guide*.

Prerequisites

This tutorial assumes that you have completed the following prerequisites:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading the AWS CLI, see Installing the AWS Command Line Interface.
- The steps in Set up to use Amazon ECS have been completed.
- Your IAM user has the required permissions specified in the AmazonECS_FullAccess IAM policy example.
- You have a VPC and security group created to use. For more information, see the section called "Create a virtual private cloud".

- The Amazon ECS CodeDeploy IAM role is created. For more information, see Amazon ECS CodeDeploy IAM Role.

Step 1: Create an Application Load Balancer

Amazon ECS services using the blue/green deployment type require the use of either an Application Load Balancer or a Network Load Balancer. This tutorial uses an Application Load Balancer.

To create an Application Load Balancer

1. Use the create-load-balancer command to create an Application Load Balancer. Specify two subnets that aren't from the same Availability Zone as well as a security group.

   ```
   aws elbv2 create-load-balancer \
        --name bluegreen-alb \
        --subnets subnet-abcd1234 subnet-abcd5678 \
        --security-groups sg-abcd1234 \
        --region us-east-1
   ```

   The output includes the Amazon Resource Name (ARN) of the load balancer, with the following format:

   ```
   arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
   e5ba62739c16e642
   ```

2. Use the create-target-group command to create a target group. This target group will route traffic to the original task set in your service.

   ```
   aws elbv2 create-target-group \
        --name bluegreentarget1 \
        --protocol HTTP \
        --port 80 \
        --target-type ip \
        --vpc-id vpc-abcd1234 \
        --region us-east-1
   ```

   The output includes the ARN of the target group, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4
```

3. Use the create-listener command to create a load balancer listener with a default rule that forwards requests to the target group.

```
aws elbv2 create-listener \
     --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
     --protocol HTTP \
     --port 80 \
     --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup
bluegreentarget1/209a844cd01825a4 \
     --region us-east-1
```

The output includes the ARN of the listener, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:listener/app/bluegreen-alb/
e5ba62739c16e642/665750bec1b03bd4
```

Step 2: Create an Amazon ECS Cluster

Use the create-cluster command to create a cluster named `tutorial-bluegreen-cluster` to use.

```
aws ecs create-cluster \
     --cluster-name tutorial-bluegreen-cluster \
     --region us-east-1
```

The output includes the ARN of the cluster, with the following format:

```
arn:aws:ecs:region:aws_account_id:cluster/tutorial-bluegreen-cluster
```

Step 3: Register a Task Definition

Use the register-task-definition command to register a task definition that is compatible with Fargate. It requires the use of the `awsvpc` network mode. The following is the example task definition used for this tutorial.

First, create a file named `fargate-task.json` with the following contents. Ensure that you use the ARN for your task execution role. For more information, see Amazon ECS task execution IAM role.

```
{
    "family": "sample-fargate",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "sample-app",
            "image": "public.ecr.aws/docker/library/httpd:latest",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
  "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
```

```
    "memory": "512"
}
```

Then register the task definition using the `fargate-task.json` file that you created.

```
aws ecs register-task-definition \
     --cli-input-json file://fargate-task.json \
     --region us-east-1
```

Step 4: Create an Amazon ECS Service

Use the create-service command to create a service.

First, create a file named `service-bluegreen.json` with the following contents.

```
{
    "cluster": "tutorial-bluegreen-cluster",
    "serviceName": "service-bluegreen",
    "taskDefinition": "tutorial-task-def",
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4",
            "containerName": "sample-app",
            "containerPort": 80
        }
    ],
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "platformVersion": "LATEST",
    "networkConfiguration": {
       "awsvpcConfiguration": {
          "assignPublicIp": "ENABLED",
          "securityGroups": [ "sg-abcd1234" ],
          "subnets": [ "subnet-abcd1234", "subnet-abcd5678" ]
       }
    },
    "desiredCount": 1
```

```
}
```

Then create your service using the `service-bluegreen.json` file that you created.

```
aws ecs create-service \
     --cli-input-json file://service-bluegreen.json \
     --region us-east-1
```

The output includes the ARN of the service, with the following format:

```
arn:aws:ecs:region:aws_account_id:service/service-bluegreen
```

Step 5: Create the AWS CodeDeploy Resources

Use the following steps to create your CodeDeploy application, the Application Load Balancer target group for the CodeDeploy deployment group, and the CodeDeploy deployment group.

To create CodeDeploy resources

1. Use the create-application command to create an CodeDeploy application. Specify the ECS compute platform.

   ```
   aws deploy create-application \
        --application-name tutorial-bluegreen-app \
        --compute-platform ECS \
        --region us-east-1
   ```

   The output includes the application ID, with the following format:

   ```
   {
       "applicationId": "b8e9c1ef-3048-424e-9174-885d7dc9dc11"
   }
   ```

2. Use the create-target-group command to create a second Application Load Balancer target group, which will be used when creating your CodeDeploy deployment group.

   ```
   aws elbv2 create-target-group \
        --name bluegreentarget2 \
        --protocol HTTP \
        --port 80 \
   ```

```
    --target-type ip \
    --vpc-id "vpc-0b6dd82c67d8012a1" \
    --region us-east-1
```

The output includes the ARN for the target group, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/708d384187a3cfdc
```

3. Use the create-deployment-group command to create an CodeDeploy deployment group.

   First, create a file named `tutorial-deployment-group.json` with the following contents. This example uses the resource that you created. For the `serviceRoleArn`, specify the ARN of your Amazon ECS CodeDeploy IAM role. For more information, see Amazon ECS CodeDeploy IAM Role.

```
{
   "applicationName": "tutorial-bluegreen-app",
   "autoRollbackConfiguration": {
      "enabled": true,
      "events": [ "DEPLOYMENT_FAILURE" ]
   },
   "blueGreenDeploymentConfiguration": {
      "deploymentReadyOption": {
         "actionOnTimeout": "CONTINUE_DEPLOYMENT",
         "waitTimeInMinutes": 0
      },
      "terminateBlueInstancesOnDeploymentSuccess": {
         "action": "TERMINATE",
         "terminationWaitTimeInMinutes": 5
      }
   },
   "deploymentGroupName": "tutorial-bluegreen-dg",
   "deploymentStyle": {
      "deploymentOption": "WITH_TRAFFIC_CONTROL",
      "deploymentType": "BLUE_GREEN"
   },
   "loadBalancerInfo": {
      "targetGroupPairInfoList": [
        {
          "targetGroups": [
             {
```

```
                "name": "bluegreentarget1"
            },
            {
                "name": "bluegreentarget2"
            }
        ],
        "prodTrafficRoute": {
            "listenerArns": [
                "arn:aws:elasticloadbalancing:region:aws_account_id:listener/
app/bluegreen-alb/e5ba62739c16e642/665750bec1b03bd4"
            ]
        }
      }
    ]
  },
  "serviceRoleArn": "arn:aws:iam::aws_account_id:role/ecsCodeDeployRole",
  "ecsServices": [
      {
          "serviceName": "service-bluegreen",
          "clusterName": "tutorial-bluegreen-cluster"
      }
  ]
}
```

Then create the CodeDeploy deployment group.

```
aws deploy create-deployment-group \
     --cli-input-json file://tutorial-deployment-group.json \
     --region us-east-1
```

The output includes the deployment group ID, with the following format:

```
{
    "deploymentGroupId": "6fd9bdc6-dc51-4af5-ba5a-0a4a72431c88"
}
```

Step 6: Create and Monitor an CodeDeploy Deployment

Use the following steps to create and upload an application specification file (AppSpec file) and an CodeDeploy deployment.

To create and monitor an CodeDeploy deployment

1. Create and upload an AppSpec file using the following steps.

   a. Create a file named `appspec.yaml` with the contents of the CodeDeploy deployment group. This example uses the resources that you created earlier in the tutorial.

   ```
   version: 0.0
   Resources:
     - TargetService:
         Type: AWS::ECS::Service
         Properties:
           TaskDefinition: "arn:aws:ecs:region:aws_account_id:task-
   definition/first-run-task-definition:7"
           LoadBalancerInfo:
             ContainerName: "sample-app"
             ContainerPort: 80
           PlatformVersion: "LATEST"
   ```

   b. Use the s3 mb command to create an Amazon S3 bucket for the AppSpec file.

   ```
   aws s3 mb s3://tutorial-bluegreen-bucket
   ```

   c. Use the s3 cp command to upload the AppSpec file to the Amazon S3 bucket.

   ```
   aws s3 cp ./appspec.yaml s3://tutorial-bluegreen-bucket/appspec.yaml
   ```

2. Create the CodeDeploy deployment using the following steps.

   a. Create a file named `create-deployment.json` with the contents of the CodeDeploy deployment. This example uses the resources that you created earlier in the tutorial.

   ```
   {
       "applicationName": "tutorial-bluegreen-app",
       "deploymentGroupName": "tutorial-bluegreen-dg",
       "revision": {
           "revisionType": "S3",
           "s3Location": {
               "bucket": "tutorial-bluegreen-bucket",
               "key": "appspec.yaml",
               "bundleType": "YAML"
           }
   ```

```
    }
}
```

b. Use the create-deployment command to create the deployment.

```
aws deploy create-deployment \
     --cli-input-json file://create-deployment.json \
     --region us-east-1
```

The output includes the deployment ID, with the following format:

```
{
    "deploymentId": "d-RPCR1U3TW"
}
```

c. Use the get-deployment-target command to get the details of the deployment, specifying the `deploymentId` from the previous output.

```
aws deploy get-deployment-target \
     --deployment-id "d-IMJU3A8TW" \
     --target-id tutorial-bluegreen-cluster:service-bluegreen \
     --region us-east-1
```

Continue to retrieve the deployment details until the status is `Succeeded`, as shown in the following output.

```
{
    "deploymentTarget": {
        "deploymentTargetType": "ECSTarget",
        "ecsTarget": {
            "deploymentId": "d-RPCR1U3TW",
            "targetId": "tutorial-bluegreen-cluster:service-bluegreen",
            "targetArn": "arn:aws:ecs:region:aws_account_id:service/service-
bluegreen",
            "lastUpdatedAt": 1543431490.226,
            "lifecycleEvents": [
                {
                    "lifecycleEventName": "BeforeInstall",
                    "startTime": 1543431361.022,
                    "endTime": 1543431361.433,
                    "status": "Succeeded"
```

```
                },
                {
                    "lifecycleEventName": "Install",
                    "startTime": 1543431361.678,
                    "endTime": 1543431485.275,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AfterInstall",
                    "startTime": 1543431485.52,
                    "endTime": 1543431486.033,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "BeforeAllowTraffic",
                    "startTime": 1543431486.838,
                    "endTime": 1543431487.483,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AllowTraffic",
                    "startTime": 1543431487.748,
                    "endTime": 1543431488.488,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AfterAllowTraffic",
                    "startTime": 1543431489.152,
                    "endTime": 1543431489.885,
                    "status": "Succeeded"
                }
            ],
            "status": "Succeeded",
            "taskSetsInfo": [
                {
                    "identifer": "ecs-svc/9223370493425779968",
                    "desiredCount": 1,
                    "pendingCount": 0,
                    "runningCount": 1,
                    "status": "ACTIVE",
                    "trafficWeight": 0.0,
                    "targetGroup": {
                        "name": "bluegreentarget1"
                    }
```

```
                },
                {
                    "identifer": "ecs-svc/9223370493423413672",
                    "desiredCount": 1,
                    "pendingCount": 0,
                    "runningCount": 1,
                    "status": "PRIMARY",
                    "trafficWeight": 100.0,
                    "targetGroup": {
                        "name": "bluegreentarget2"
                    }
                }
            ]
        }
    }
}
```

Step 7: Clean Up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using.

Cleaning up the tutorial resources

1. Use the delete-deployment-group command to delete the CodeDeploy deployment group.

```
aws deploy delete-deployment-group \
     --application-name tutorial-bluegreen-app \
     --deployment-group-name tutorial-bluegreen-dg \
     --region us-east-1
```

2. Use the delete-application command to delete the CodeDeploy application.

```
aws deploy delete-application \
     --application-name tutorial-bluegreen-app \
     --region us-east-1
```

3. Use the delete-service command to delete the Amazon ECS service. Using the `--force` flag allows you to delete a service even if it has not been scaled down to zero tasks.

```
aws ecs delete-service \
```

```
    --service arn:aws:ecs:region:aws_account_id:service/service-bluegreen \
    --force \
    --region us-east-1
```

4. Use the delete-cluster command to delete the Amazon ECS cluster.

```
aws ecs delete-cluster \
     --cluster tutorial-bluegreen-cluster \
     --region us-east-1
```

5. Use the s3 rm command to delete the AppSpec file from the Amazon S3 bucket.

```
aws s3 rm s3://tutorial-bluegreen-bucket/appspec.yaml
```

6. Use the s3 rb command to delete the Amazon S3 bucket.

```
aws s3 rb s3://tutorial-bluegreen-bucket
```

7. Use the delete-load-balancer command to delete the Application Load Balancer.

```
aws elbv2 delete-load-balancer \
     --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
     --region us-east-1
```

8. Use the delete-target-group command to delete the two Application Load Balancer target groups.

```
aws elbv2 delete-target-group \
     --target-group-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4 \
     --region us-east-1
```

```
aws elbv2 delete-target-group \
     --target-group-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/708d384187a3cfdc \
     --region us-east-1
```

Amazon ECS service quotas

When you create your AWS account, we set default *quotas* (also referred to as limits) on your AWS resources on a per-Region basis. If you attempt to exceed the quota for a resource, the request fails.

The Service Quotas console is a central location where you can view and manage your quotas for AWS services, and request a quota increase for many of the resources that you use. Use the quota information that we provide to manage your resources. Plan to request any quota increases in advance of the time that you'll need them.

For more information about Amazon ECS and Fargate quotas, see [Amazon ECS endpoints and quotas]() in the *Amazon Web Services General Reference*. For information about API throttling in the Amazon ECS API, see [Request throttling for the Amazon ECS API]().

Managing your Amazon ECS and AWS Fargate service quotas in the AWS Management Console

Amazon ECS has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see [What is Service Quotas?]() in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of your Amazon ECS service quotas.

AWS Management Console

To view Amazon ECS and Fargate service quotas using the AWS Management Console

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Elastic Container Service (Amazon ECS)** or **AWS Fargate**.

   In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.
4. To view additional information about a service quota, such as the description, choose the quota name.

5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the AWS Management Console see the Service Quotas User Guide. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

AWS CLI

To view Amazon ECS and Fargate service quotas using the AWS CLI

Run the following command to view the default Amazon ECS quotas.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code ecs \
    --output table
```

Run the following command to view the default Fargate quotas.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code fargate \
    --output table
```

Run the following command to view your applied Fargate quotas.

```
aws service-quotas list-service-quotas \
    --service-code fargate
```

Note

Amazon ECS doesn't support applied quotas.

For more information about working with service quotas using the AWS CLI, see the Service Quotas AWS CLI Command Reference. To request a quota increase, see the `request-service-quota-increase` command in the AWS CLI Command Reference.

Handle Amazon ECS service quotas and API throttling limits

Amazon ECS is integrated with several AWS services, including Elastic Load Balancing, AWS Cloud Map, and Amazon EC2. With this tight integration, Amazon ECS includes several features such as service load balancing, Service Connect, task networking, and cluster auto scaling. Amazon ECS and the other AWS services that it integrates with all maintain service quotas and API rate limits to ensure consistent performance and utilization. These service quotas also prevent the accidental provisioning of more resources than needed and protect against malicious actions that might increase your bill.

By familiarizing yourself with your service quotas and the AWS API rate limits, you can plan for scaling your workloads without worrying about unexpected performance degradation. For more information, see [Request throttling for the Amazon ECS API](#).

When scaling your workloads on Amazon ECS, we recommend that you consider the following service quota.

- AWS Fargate has quotas that limit the number of concurrent running tasks in each AWS Region. There are quotas for both On-Demand and Fargate Spot tasks on Amazon ECS. Each service quota also includes any Amazon EKS pods that you run on Fargate.
- For tasks that run on Amazon EC2 instances, the maximum number of Amazon EC2 instances that you can register for each cluster is 5,000. If you use Amazon ECS cluster auto scaling with an Auto Scaling group capacity provider, or if you manage Amazon EC2 instances for your cluster on your own, this quota might become a deployment bottleneck. If you require more capacity, you can create more clusters or request a service quota increase.
- If you use Amazon ECS cluster auto scaling with an Auto Scaling group capacity provider, when scaling your services consider the `Tasks in the PROVISIONING state per cluster` quota. This quota is the maximum number of tasks in the `PROVISIONING` state for each cluster for which capacity providers can increase capacity. When you launch a large number of tasks all at the same time, you can easily meet this quota. One example is if you simultaneously deploy tens of services, each with hundreds of tasks. When this happens, the capacity provider needs to launch new container instances to place the tasks when the cluster has insufficient capacity. While the capacity provider is launching additional Amazon EC2 instances, the Amazon ECS service scheduler likely will continue to launch tasks in parallel. However, this activity might be throttled because of insufficient cluster capacity. The Amazon ECS service scheduler implements a back-off and exponential throttling strategy for retrying task placement as new container instances are launched. As a result, you might experience slower deployment or scale-out times.

To avoid this situation, you can plan your service deployments in one of the following ones. Either deploy a large number of tasks don't require increasing cluster capacity, or keep spare cluster capacity for new task launches.

In addition to considering Amazon ECS service quota when scaling your workloads, consider also the service quota for the other AWS services that are integrated with Amazon ECS.

Elastic Load Balancing

You can configure your Amazon ECS services to use Elastic Load Balancing to distribute traffic evenly across the tasks. For more information and recommended best practices for how to choose a load balancer, see Use load balancing to distribute Amazon ECS service traffic.

Elastic Load Balancing service quotas

When you scale your workloads, consider the following Elastic Load Balancing service quotas. Most Elastic Load Balancing service quotas are adjustable, and you can request an increase in the Service Quotas console.

Application Load Balancer

When you use an Application Load Balancer, depending on your use case, you might need to request a quota increase for:

- The `Targets per Application Load Balancer` quota which is the number of targets behind your Application Load Balancer.
- The `Targets per Target Group per Region` quota which is the number of targets behind your Target Groups.

For more information, see Quotas for your Application Load Balancers in *User Guide for Application Load Balancers*.

Network Load Balancer

There are stricter limitations on the number of targets you can register with a Network Load Balancer. When using a Network Load Balancer, you often will want to enable cross-zone support, which comes with additional scaling limitations on `Targets per Availability Zone Per Network Load Balancer` the maximum number of targets per Availability Zone for each

Network Load Balancer. For more information, see Quotas for your Network Load Balancers iin the *User Guide for Network Load Balancers*.

Elastic Load Balancing API throttling

When you configure an Amazon ECS service to use a load balancer, the target group health checks must pass before the service is considered healthy. For performing these health checks, Amazon ECS invokes Elastic Load Balancing API operations on your behalf. If you have a large number of services configured with load balancers in your account, you might slower service deployments because of potential throttling specifically for the `RegisterTarget`, `DeregisterTarget`, and `DescribeTargetHealth` Elastic Load Balancing API operations. When throttling occurs, throttling errors occur in your Amazon ECS service event messages.

If you experience AWS Cloud Map API throttling, you can contact Support for guidance on how to increase your AWS Cloud Map API throttling limits. For more information about monitoring and troubleshooting such throttling errors, see Handle Amazon ECS throttling issues.

Elastic network interfaces

With your tasks use the `awsvpc` network mode, Amazon ECS provisions a unique elastic network interface (ENI) for each task. When your Amazon ECS services use an Elastic Load Balancing load balancer, these network interfaces are also registered as targets to the appropriate target group defined in the service.

Elastic network interface service quotas

When you run tasks that use the `awsvpc` network mode, a unique elastic network interface is attached to each task. If those tasks must be reached over the internet, assign a public IP address to the elastic network interface for those tasks. When you scale your Amazon ECS workloads, consider these two important quotas:

- The `Network interfaces per Region` quota which is the maximum number of network interfaces in an AWS Region for your account.
- The `Elastic IP addresses per Region` quota which is the maximum number of elastic IP addresses in an AWS Region.

Both of these service quotas are adjustable and you can request an increase from your Service Quotas console for these. For more information, see Amazon VPC service quotas in the *Amazon Virtual Private Cloud user Guide*.

For Amazon ECS workloads that are hosted on Amazon EC2 instances, when running tasks that use the `awsvpc` network mode consider the `Maximum network interfaces` service quota, the maximum number of network instances for each Amazon EC2 instance. This quota limits the number of tasks that you can place on an instance. You cannot adjust the quota and it's not available in the Service Quotas console. For more information, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide*.

Although you can't change the number of network interfaces that can be attached to an Amazon EC2 instance, you can use the elastic network interface trunking feature to increase the number of available network interfaces. For example, by default a `c5.large` instance can have up to three network interfaces. The primary network interface for the instance counts as one. So, you can attach an additional two network interfaces to the instance. Because each task that uses the `awsvpc` network mode requires a network interface, you can typically only run two such tasks on this instance type. This can lead to under-utilization of your cluster capacity. If you enable elastic network interface trunking, you can increase the network interface density to place a larger number of tasks on each instance. With trunking turned on, a `c5.large` instance can have up to 12 network interfaces. The instance has the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the instance. As a result, with this configuration you can run 10 tasks on the instance instead of the default two tasks. For more information, see Increasing Amazon ECS Linux container instance network interfaces.

Elastic network interface API throttling

When you run tasks that use the `awsvpc` network mode, Amazon ECS relies on the following Amazon EC2 APIs. Each of these APIs have different API throttles. For more information, see Request throttling for the Amazon EC2 API in the *Amazon EC2 API Reference*.

- CreateNetworkInterface
- AttachNetworkInterface
- DetachNetworkInterface
- DeleteNetworkInterface
- DescribeNetworkInterfaces
- DescribeVpcs
- DescribeSubnets
- DescribeSecurityGroups
- DescribeInstances

If the Amazon EC2 API calls are throttled during the elastic network interface provisioning workflows, the Amazon ECS service scheduler automatically retries with exponential back-offs. These automatic retires might sometimes lead to a delay in launching tasks, which results in slower deployment speeds. When API throttling occurs, you will see the message `Operations are being throttled. Will try again later.` on your service event messages. If you consistently meet Amazon EC2 API throttles, you can contact Support for guidance on how to increase your API throttling limits. For more information about monitoring and troubleshooting throttling errors, see Handling throttling issues.

AWS Cloud Map

Amazon ECS service discovery and Service Connect use AWS Cloud Map APIs to manage namespaces for your Amazon ECS services. If your services have a large number of tasks, consider the following recommendations.

AWS Cloud Map service quotas

When Amazon ECS services are configured to use service discovery or Service Connect, the `Tasks per service` quota which is the maximum number of tasks for the service, is affected by the AWS Cloud Map `Instances per service` service quota which is the maximum number of instances for that service. In particular, the AWS Cloud Map service quota decreases the amount of instances that you can run to at most 1,000 instances per service. You cannot change the AWS Cloud Map quota. For more information, see AWS Cloud Map service quotas.

AWS Cloud Map API throttling

Amazon ECS calls the `ListInstances`, `GetInstancesHealthStatus`, `RegisterInstance`, and `DeregisterInstance` AWS Cloud Map APIs on your behalf. They help with service discovery and perform health checks when you launch a task. When multiple services that use service discovery with a large number of tasks are deployed at the same time, this can result in exceeding the AWS Cloud Map API throttling limits. When this happens, you likely will see the following message: `Operations are being throttled. Will try again later` in your Amazon ECS service event messages and slower deployment and task launch speed. AWS Cloud Map doesn't document throttling limits for these APIs. If you experience throttling from these, you can contact Support for guidance on increasing your API throttling limits. For more recommendations about monitoring and troubleshooting such throttling errors, see Handle Amazon ECS throttling issues.

Amazon ECS API reference

In addition to the AWS Management Console and the AWS Command Line Interface (AWS CLI), Amazon ECS also provides an API. You can use the API to automate tasks for managing Amazon ECS resources.

- For a list of API operations by Amazon ECS resource, see Actions by Amazon ECS resource.
- For an alphabetical list of API operations, see Actions.
- For an alphabetical list of data types, see Data types.
- For a list of common query parameters, see Common parameters.
- For descriptions of the error codes, see Common errors.

For more information about the AWS CLI, see AWS Command Line Interface reference for Amazon ECS.

Document history

The following table describes the major updates and new features for the *Amazon Elastic Container Service Developer Guide*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
Support for running ECS Exec commands from the console	You can now run ECS Exec commands directly from the Amazon ECS console. For more information, see Running ECS Exec commands from the console.	September 4, 2025
Add permissions to AmazonEC2ContainerServiceforEC2Role managed policy.	The `AmazonEC2ContainerServiceforEC2Role` managed IAM policy was updated to include the `ecs:ListTagsForResource` permission. This permission allows the Amazon ECS agent to retrieve task and container instance tags through the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`). For more information, see AmazonEC2ContainerServiceforEC2Role.	August 4, 2025
Add permissions to AmazonECSInfrastructureRolePolicyForLoadBalancers.	The `AmazonECSInfrastructureRolePolicyForLoadBalancers` managed IAM policy was updated with new permissions for describing, deregistering and registering target groups.	July 25, 2025
Support for updating a service deployment controller	Amazon ECS supports updating a service deployment controller, allowing you to change between different deployment types for an existing service. For more information, see Update Amazon ECS service parameters.	July 15, 2025
Amazon ECS native blue/green deployments	Amazon ECS supports native blue/green deployments, allowing you to validate new service revisions before directing production traffic to them. For more	July 15, 2025

Change	Description	Date
	information, see Amazon ECS blue/green deployments.	
Add new AmazonECSInfrastructureRolePolicyForLoadBalancers managed policy.	This policy provides access to other AWS service resources required to manage load balancers associated with Amazon ECS workloads on your behalf. For more information, see AmazonECSInfrastructureRolePolicyForLoadBalancers.	July 15, 2025
SOCI index manifest v2 support	Fargate customers looking to use SOCI will only be able to use the SOCI index manifest v2. Existing customers that have previously used SOCI on Fargate can continue to use the SOCI index manifest v1, however we strongly advise those customers migrate to SOCI index manifest v2. For more information, see Lazy loading container images using Seekable OCI (SOCI).	July 2, 2025
Added the task ID in service action events generated due to health failures.	For more information, see Amazon ECS service unhealthy event messages.	June 30, 2025
Update to default log driver mode.	The default logging mode for Amazon ECS has been updated from `blocking` to `non-blocking` . You can override this default mode by specifying a `mode` in your container definition's `logConfiguration` or configure your own default by setting the `defaultLogDriverMode` account setting using the Amazon ECS console or the AWS CLI. For more information, see Default log driver mode.	June 25, 2025

Change	Description	Date
Add permissions to AmazonECSServiceRolePolicy.	The `AmazonECSServiceRolePolicy` managed IAM policy was updated with new AWS Cloud Map permissions which Amazon ECS can update AWS Cloud Map service attributes for services that Amazon ECS manages.	June 24, 2025
Support for the Amazon ECS-optimized Windows Server 2025 Full AMI and Amazon ECS-optimized Windows Server 2025 Core AMI.	For more information, see Amazon ECS-optimized Windows AMIs.	June 20, 2025
Support for updating a service capacity provider strategy.	You can now update a service capacity provider strategy. For more information, see Update service parameters.	June 12, 2025
Add permissions to AmazonECSInfrastructureRolePolicyForVolumes	The `AmazonECSInfrastructureRolePolicyForVolumes` policy has been updated to add the `ec2:DescribeInstances` permission. The permission helps prevent device name collision for Amazon EBS volumes that are attached to Amazon ECS tasks that run on the same container instance.	June 2, 2025
Support for volume initialization rate for Amazon EBS volumes attached to Amazon ECS tasks.	You can now set a `volumeIntiializationRate` to determine the rate, in MiB/s, at which data is fetched from a snapshot of an existing Amazon EBS volume to create new volumes for attachment to Amazon ECS tasks. For more information, see volumeInitializationRate.	May 13, 2025
Support for stopping a service deployment.	You can now stop a service deployment. For more information, see Stopping Amazon ECS service deployments.	May 2, 2025

Change	Description	Date
Support for Amazon ECS dual-stack endpoints.	You can now use dual-stack endpoints to interact with Amazon ECS. Dual-stack endpoints support requests to Amazon ECS over both Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). For more information, see Using Amazon ECS dual-stack endpoints.	April 30, 2025
Support for configuring a default log driver mode at the account level in an AWS Region.	You can now configure a default delivery mode of log messages from a container to the log driver that you specify in your `logConfiguration` using the `defaultLogDriverMode` account setting. For more information, see Default log driver mode.	April 17, 2025
Support for migrating a service with a short ARN format to the long ARN format.	You can now migrate a service with a short ARN format to the long ARN formatk packet. For more information, see Migrate an Amazon ECS short service ARN to a long ARN.	February 13, 2025
Support for three types of fault injection on Amazon ECS and Fargate.	You can now use network blackhole port, network latency, and network packet loss fault injection with Amazon ECS and Fargate. For more information, see Use AWS Fault Injection Service to test your Amazon ECS workloads.	December 19, 2024
Support for CloudWatch Container Insights with enhanced observability.	You can use CloudWatch Container Insights with enhanced observability. For more information, see Monitor Amazon ECS containers using Container Insights with enhanced observability.	December 2, 2024
Support for predictive scaling.	You can use predictive scaling to anticipate future needs and proactively increase tasks in your service as needed. For more information, see Use historical patterns to scale CloudWatch services with predictive scaling.	November 21, 2024

Change	Description	Date
Support for service-rebalancing.	You can now have Amazon ECS automatically balance your service tasks across Availability Zones. For more information, see Balancing an Amazon ECS service across Availability Zones.	November 19, 2024
Support for configuring resolution of container image tags to image digests.	You can now configure the resolution of container image tags to digests for each container in tasks that are part of a service. For more information, see Container image resolution and `versionConsistency`.	November 19, 2024
Add new `AmazonECSInfrastructureRolePolicyForVpcLattice` IAM policy	The `AmazonECSInfrastructureRolePolicyForVpcLattice` policy provides access to other AWS service resources required to manage VPC Lattice feature in Amazon ECS workloads on your behalf.	November 18, 2024
VPC Lattice is now available for Amazon ECS services.	You can now use VPC Lattice to standardize your service-to-service connectivity, security, and observability. For more information, see Use Amazon VPC Lattice to connect, observe, and secure your Amazon ECS services.	November 18, 2024
Support for service history using service deployments and service revisions.	You can use service deployments and service revisions to view deployment history. For more information, see View service history using Amazon ECS service deployments and Amazon ECS service revisions.	October 30, 2024
Support for Amazon EBS volumes for the EC2 launch type using the Windows operating system.	Amazon ECS support Amazon EBS volumes attached to tasks running on EC2 instances with the Windows operating system. For more information, see Use Amazon EBS volumes with Amazon ECS.	October 24, 2024

Change	Description	Date
Add permissions to AmazonECSInfrastructureRolePolicyForVolumes.	The `AmazonECSInfrastructureRolePolicyForVolumes` policy has been updated to allow customers to create an Amazon EBS volume from a snapshot. For more information, see AmazonECSInfrastructureRolePolicyForVolumes.	October 10, 2024
Fargate Spot is available for Linux ARM.	Support for Fargate Spot on Linux ARM was added to Amazon ECS.	September 6, 2024
Container restart policy for individual containers in Amazon ECS tasks.	You can enable restart policies for essential and non-essential containers defined in your task definition to overcome transient failures faster and maintain task availability. For more information, see Restart individual containers in Amazon ECS tasks with container restart policies.	August 15, 2024
Added permissions to the section called "AmazonECS_FullAccess"	The `AmazonECS_FullAccess` policy was updated to add `iam:PassRole` permissions for IAM roles for a role named `ecsInfrastructureRole`. This is the default IAM role created by the AWS Management Console that is intended to be used as an ECS infrastructure role that allows Amazon ECS to manage Amazon EBS volumes attached to ECS tasks.	August 13, 2024
Amazon Linux 2023, CentOS Stream 9, Fedora 40, and Ubuntu 24 support added for ECS Anywhere	Support for the Amazon Linux 2023, CentOS Stream 9, Fedora 40, and Ubuntu 24 operating systems was added to ECS Anywhere. For more information, see Supported operating systems and system architectures.	July 23, 2024
Container image resolution for services that use the *rolling update (ECS)* deployment type.	To ensure that all tasks in a service that uses the rolling update deployment controller use the same container image, Amazon ECS resolves container image names and any image tags specified in the task definition to container image digests. For more information, see Container image resolution.	July 10, 2024

Change	Description	Date
Debian 11 and Debian 12 support added for ECS Anywhere	Support for the Debian 11 and Debian 12 operating systems was added to ECS Anywhere. For more information, see Supported operating systems and system architectures.	March 28, 2024
gMSA for Linux Containers on Fargate support	Amazon ECS supports Active Directory authentication for Linux containers on Fargate through a special kind of service account called a group Managed Service Account (gMSA). For more information, see Using gMSA for Linux containers on Fargate.	March 5, 2024
CloudWatch metrics added for Amazon EBS volumes attached to tasks	Amazon ECS now publishes CloudWatch metrics for the Amazon EBS storage utilization for tasks that have an Amazon EBS volume attached. For more information, see Amazon ECS CloudWatch metrics.	February 8, 2024
Service Connect TLS	You can now use TLS with Service Connect.	January 22, 2024
Service Connect TLS managed policy	Added new AmazonECSInfrastructureRolePolicyForServiceConnectTransportLayerSecurity policy.	January 22, 2024
Service Connect timeout configuration update	Service Connect timeout configuration can now be updated and includes two optional parameters - `idleTimeout` and `perRequestTimeout` .	January 22, 2024
Amazon ECS managed instance draining	You can use Amazon ECS managed instance draining to facilitate graceful termination of Amazon ECS instances.	January 19, 2024
Ubuntu 22 support added for ECS Anywhere	Support for the Ubuntu 22 operating system was added to ECS Anywhere. For more information, see Supported operating systems and system architectures.	January 16, 2024

Change	Description	Date
Add `AmazonECSInfrastructureRolePolicyForVolumes` IAM policy	The `AmazonECSInfrastructureRolePolicyForVolumes` was added. The policy grants the permissions that are needed by Amazon ECS to make AWS API calls to manage Amazon EBS volumes associated with Amazon ECS workloads.	January 11, 2024
Amazon EBS data volume for Amazon ECS task	You can configure 1 Amazon EBS data volume per task during deployment for attachment to standalone Amazon ECS tasks or tasks managed by an ECS service. Configuring a volume at deployment allows you to create resuable task definitions not constrained to specific volume types or settings. Amazon EBS volumes provide a highly available, cost-effective, durable, high-performance block storage for data intensive containerized workloads.	January 11, 2024
Amazon ECS classic console reached end of life	The Amazon ECS console has reached the end of life.	December 4, 2023
Updated policy	The AmazonECSServiceRolePolicy managed IAM policy was updated with new `events` permissions and additional `autoscaling` and `autoscaling-plans` permissions.	December 4, 2023
Runtime Monitoring support	You can use Runtime Monitoring to monitor your Amazon ECS workloads to identify malicious or unauthorized behavior. For more information, see Runtime Monitoring.	November 26, 2023
Updated policy	The `AmazonECSServiceRolePolicy` managed IAM policy was updated to allow access to the AWS Cloud Map `DiscoverInstancesRevision` API.	October 4, 2023

Change	Description	Date
AWS Fargate task retirement configuration	You can configure the wait period before Fargate tasks are retired For more information, see AWS Fargate task maintenance.	September 5, 2023
Additional task definition parameters in AWS Fargate	AWS Fargate adds support for `pidMode` and `systemControls` in Linux platform version `1.4.0`. For more information, see Task definitions.	August 9, 2023
Amazon ECS console task definition page redesign	The task definition page in the Amazon ECS console has been redesigned and contains additional options. For more information, see Creating a task definition using the console.	July 26, 2023
Fargate supports lazy loading with Seekable OCI indexes	AWS Fargate is introducing Seekable OCI (SOCI) indexes. With SOCI, containers only spend a few seconds on the image pull before they can start, providing time for environment setup and application instantiation while the image is downloaded in the background. For more information, see Lazy loading container images using Seekable OCI (SOCI).	July 17, 2023
Improved support for gMSA on Linux and Windows	The task definition has a new `credentialSpecs` field for gMSA for Linux and Windows. A new complete tutorial for domainless gMSA on Windows has been added, see Tutorial: Using Windows Containers with Domainless gMSA using the AWS CLI. For more information, see Using gMSAs for Linux Containers and Using gMSAs for Windows Containers.	July 14, 2023

Change	Description	Date
Improved ECS Agent versions documentation	The documentation for the Amazon ECS Agent versions has been updated. We recommend that you use the `v20.10.13` version or newer of Docker with the latest version of the Amazon ECS container agent. The released versions and changes to the agent are available on GitHub. For more information, see Amazon ECS Linux container agent versions.	June 20, 2023
Updated Region availability for Fargate ARM64 support	The Region availability for Fargate ARM64 support has been updated. For more information, see Considerations.	June 19, 2023
Improve cluster auto scaling documentation	The documentation for Amazon ECS scaling of Amazon EC2 Auto Scaling has significant improvements in accuracy and readability. For more information, see Amazon ECS cluster auto scaling.	May 4, 2023
Tagging authorization for resource creation.	Users must have permissions for actions that create the resource, such as `ecsCreateCluster`. When you create a resource and specify tags for that resource, AWS performs additional authorization to verify that there are permissions to create tags. For more information, see Tagging authorization and Grant permission to tag resources on creation.	April 18, 2023
Support for gMSA for Linux containers on EC2	You can use gMSA to authenticate to Active Directory for Linux containers on EC2. For more information, see Using gMSAs for Linux Containers.	April 14, 2023
Support for ephemeral storage for Windows containers on AWS Fargate	You can use ephemeral storage for Windows containers on AWS Fargate. For more information, see Fargate task storage.	April 14, 2023

Change	Description	Date
AWS Cost Management support for task-level CUR data	You can turn on task-level cost and resource usage in the Cost and Usage Reports. This adds Split Cost Allocation Data for tasks that run on AWS Fargate and EC2. For more information, see Task-level Cost and Usage Reports.	April 12, 2023
Amazon Linux 2023 Amazon ECS-optimized AMI	You can deploy workloads on the Amazon Linux 2023 Amazon ECS-optimized AMI. For more information, see Amazon ECS-optimized Linux AMIs.	April 10, 2023
AWS Fargate Federal Information Processing Standard (FIPS) 140	You can deploy workloads on Amazon ECS on AWS Fargate in a manner compliant with Federal Information Processing Standard (FIPS) 140. For more information, see AWS Fargate Federal Information Processing Standard (FIPS-140).	April 10, 2023
Task definition deletion	You can delete a task definition using the Amazon ECS console, SDK, and AWS CLI. For more information, see Deleting a task definition revision using the console and Task definitions.	February 24, 2023
AWS Fargate service recommendations in Compute Optimizer	AWS Compute Optimizer generates task and container size recommendations based on the utilization of running tasks in Amazon ECS services on AWS Fargate. For more information, see Viewing recommendations for Amazon ECS services on Fargate.	January 27, 2023
Amazon ECS console	The new Amazon ECS console is now the default console. For more information, see New Amazon ECS console.	January 19, 2023
Updated `AmazonECS_FullAccess` IAM policy	The `AmazonECS_FullAccess` IAM policy is updated to include permissions to add tags to load balancers during creation. For more information, see AmazonECS_FullAccess.	January 4, 2023

Change	Description	Date
Use CloudWatch alarms to detect Amazon ECS service deployment failures	You can configure Amazon ECS to set the deploymen t to failed when it detects that a specified CloudWatc h alarm has gone into the ALARM state. For more information, see the section called "Rolling update deployments".	December 19, 2022
Support for container port mapping	You can set a port number range on the container that's bound to the dynamically mapped host port range. For more information, see the section called "Port mappings".	December 15, 2022
General availability of Amazon ECS Service Connect	This feature adds service discovery and service mesh that is controlled by Amazon ECS service deploymen ts. For more information, see the section called " Service Connect".	November 27, 2022
The new Amazon ECS console experience for task definitions is updated	The new Amazon ECS console experience now contains a JSON editor for task defintions. For more information, see the section called "Creating a task definition using the console".	October 27, 2022
The new Amazon ECS console experience for task definitions is updated	The new Amazon ECS console experience now contains a JSON editor for task defintions. For more information, see the section called "Creating a task definition using the console".	October 27, 2022
The new Amazon ECS console experience is updated	The new Amazon ECS console experience has been updated with additional service and task parameter s. For more information, see the section called "Creating a rolling update deployment" and the section called "Running an application as a task".	October 7, 2022
New information in task metadata endpoint version 4	The task metadata endpoint version 4 now invludes the VPC ID and the service name. For more informati on, see the section called "Task metadata endpoint version 4".	October 7, 2022

Change	Description	Date
New task definition sizes	Amazon ECS on Fargate now supports the 8 vCPU and 16 vCPU task sizes. For more information, see the section called "Task size".	September 16, 2022
ECS CLI pages archived	The ECS CLI documentation has been archived. We recommend using AWS Copilot for your command line tool needs. For more information, see Creating Amazon ECS resources using the AWS Copilot command line interface.	September 15, 2022
New Fargate quotas	Fargate is transitioning from task count-based quotas to vCPU-based quotas. For more information, see Amazon ECS endpoints and quotas.	September 8, 2022
Support for Amazon EC2 Auto Scaling warm pools.	You can now use Amazon EC2 Auto Scaling warm pools to scale out your applications faster and save costs. For more information, see Configuring pre-initialized instances for your Amazon ECS Auto Scaling group.	March 23, 2022
Support for Windows instances in ECS Anywhere.	ECS Anywhere now supports Windows instances. For more information, see Amazon ECS clusters for the external launch type.	March 3, 2022
Added ECS Exec support for external instances	ECS Exec is now supported for external instances . For more information, see Monitor Amazon ECS containers with ECS Exec.	January 24, 2022

Change	Description	Date
The new Amazon ECS console experience updated	The new Amazon ECS console experience supports creating and deleting a cluster, updating a task definition, and deregistering a task definition. For more information, see Creating an Amazon ECS cluster for the Fargate launch type, Deleting an Amazon ECS cluster, Updating an Amazon ECS task definition using the console, and Deregistering an Amazon ECS task definition revision using the console.	December 8, 2021
The new Amazon ECS console experience updated	The new Amazon ECS console experience supports creating a task definition. For more information, see Creating an Amazon ECS task definition using the console.	November 23, 2021
Amazon ECS supports the 64-bit ARM architecture for Linux.	Amazon ECS supports the 64-bit ARM CPU architecture for the Linux operating system. For more information, see the section called "Task definitions for 64-bit ARM workloads".	November 23, 2021
Amazon ECS support for the fluentd log-driver-buffer-limit option	Amazon ECS supports the fluentd `log-driver-buffer-limit` option. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.	November 22, 2021
Amazon ECS-optimized Linux AMI build script	Amazon ECS has open-sourced the build scripts that are used to build the Linux variants of the Amazon ECS-optimized AMI. For more information, see Amazon ECS-optimized Linux AMI build script.	November 19, 2021
Container instance health	Amazon ECS adds support for container instance health monitoring. For more information, see Monitor Amazon ECS container instance health.	November 10, 2021

Change	Description	Date
Support for Windows Amazon ECS Exec	Amazon ECS Exec supports Windows. For more information, see Monitor Amazon ECS containers with ECS Exec.	November 1, 2021
Support for Windows containers on Fargate.	Amazon ECS supports Windows containers on Fargate. For more information, see Fargate Windows platform version change log.	October 28, 2021
GPU support for external instances on Amazon ECS Anywhere	Amazon ECS supports specifying GPU requireme nts in the task definition for tasks run on external instances. For more information, see Amazon ECS task definitions for GPU workloads and Registering an external instance to an Amazon ECS cluster.	October 8, 2021
Support of `awsvpc` network mode on Windows	Amazon ECS supports `awsvpc` network mode on Windows. For more information, see Allocate a network interface for an Amazon ECS task.	July 15, 2021
General availability of Bottlerocket	Amazon ECS supports an Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI. For more information, see Amazon ECS-optimized Bottlerocket AMIs.	June 30, 2021
Amazon ECS scheduled tasks update	Amazon EventBridge added support for additional parameters when creating rules that trigger Amazon ECS scheduled tasks.	June 25, 2021
AWS managed policies for Amazon ECS	Amazon ECS added documentation of AWS managed policies for service-linked roles. For more informati on, see AWS managed policies for Amazon Elastic Container Service.	June 8, 2021
Getting started with the AWS CDK	Added a getting started guide for using the AWS CDK with Amazon ECS. For more information, see Creating Amazon ECS resources using the AWS CDK.	May 27, 2021

Change	Description	Date
Amazon ECS Anywhere	Amazon ECS has added support for registering an on-premise server or virtual machine (VM) with your cluster. For more information, see Amazon ECS clusters for the external launch type.	May 25, 2021
Amazon ECS-optimized Windows Server 20H2 Core AMI	Amazon ECS has added support for a new Windows Amazon ECS-optimized AMI variant for Windows Server 20H2 Core. For more information, see Amazon ECS-optimized Linux AMIs.	April 19, 2021
Amazon ECS Exec	Amazon ECS has released a new debugging tool called ECS Exec. For more information, see Monitor Amazon ECS containers with ECS Exec.	March 15, 2021
VPC endpoint policy support	Amazon ECS now supports VPC endpoint policies. For more information, see Creating a VPC endpoint policy for Amazon ECS.	January 11, 2021
New console experience	Amazon ECS has released a new console experienc e which supports creating or updating a service or running a standalone task. For more information, see Creating an Amazon ECS rolling update deployment and Running an application as an Amazon ECS task.	December 28, 2020
Capacity provider update	Amazon ECS added support for updating an existing Auto Scaling group capacity provider.	November 23, 2020
ECS now supporting Amazon FSx for Windows File Server for Windows tasks	Amazon ECS added support for specifying Amazon FSx for Windows File Server volumes in Windows task definitions. For more information, see Use FSx for Windows File Server volumes with Amazon ECS.	November 11, 2020

Change	Description	Date
VPC dual-stack mode support added	Amazon ECS added support for using a VPC in dual-stack mode with tasks using the `awsvpc` network mode, which provides support for IPv6 addresses. For more information, see Using a VPC in dual-stack mode.	November 5, 2020
Task metadata endpoint v4 update	Amazon ECS added additional metadata to the task metadata endpoint v4 output. For more information, see Amazon ECS task metadata endpoint version 4.	November 5, 2020
Support for Local Zones and Wavelength Zones	Amazon ECS added support for workloads in Local Zones and Wavelength Zones. For more information, see Amazon ECS applications in shared subnets, Local Zones, and Wavelength Zones.	September 4, 2020
Amazon ECS variant of Bottlerocket AMI	Bottlerocket is a Linux-based open source operating system that is purpose-built by AWS for running containers. An Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI you can use when launching Amazon ECS container instances. For more information, see Amazon ECS-optimized Bottlerocket AMIs.	August 31, 2020
Task metadata endpoint version 4 updated for network rate stats	The task metadata endpoint version 4 has been updated to provide network rate stats for Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 instances running at least version `1.43.0` of the container agent. For more information, see Amazon ECS task metadata endpoint version 4.	August 10, 2020
Fargate usage metrics	AWS Fargate provides CloudWatch usage metrics which provide visibility into your accounts usage of Fargate On-Demand and Fargate Spot resources. For more information, see Usage metrics.	August 3, 2020

Change	Description	Date
AWS Copilot version 0.1.0	The new AWS Copilot CLI launched, providing high-level commands to simplify modeling, creating, releasing, and managing containerized applications on Amazon ECS from a local development environment. For more information, see Creating Amazon ECS resources using the AWS Copilot command line interface.	July 9, 2020
AWS Fargate platform versions deprecation schedule	The Fargate platform version deprecation schedule has been added. For more information, see AWS Fargate Linux platform version deprecation.	July 8, 2020
AWS Fargate Region expansion	Amazon ECS on AWS Fargate has expanded to the Europe (Milan) Region.	June 25, 2020
Amazon ECS optimized Amazon Linux 2 (Neuron) AMI released	Amazon ECS released an Amazon ECS optimized Amazon Linux 2 (Neuron) AMI for inferential workloads. For more information, see Amazon ECS-optimized Linux AMIs.	June 24, 2020
Added support for deleting capacity providers	Amazon ECS added support for deleting Auto Scaling group capacity providers.	June 11, 2020
AWS Fargate platform version 1.4.0 update	Beginning on May 28, 2020, any new Fargate task that is launched using platform version 1.4.0 will have its 20 GB ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key. For more information, see Fargate task ephemeral storage for Amazon ECS.	May 28, 2020

Change	Description	Date
Environment variable file support	Added support for specifying environment variable files in a task definition, which enables you to bulk add environment variables to your containers. For more information, see Pass an individual environment variable to an Amazon ECS container.	May 18, 2020
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Africa (Cape Town) Region.	May 11, 2020
Service quota updated	The following service quota was updated: • Clusters per account was raised from `2,000` to `10,000`. For more information, see Amazon ECS endpoints and quotas.	April 17, 2020

Change	Description	Date
AWS Fargate platform version 1.4.0	AWS Fargate platform version 1.4.0 is released, which contains the following features: • Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see Use Amazon EFS volumes with Amazon ECS. • The ephemeral task storage has been increased to 20 GB. For more information, see Fargate task ephemeral storage for Amazon ECS. • The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see Amazon ECS task networking options for the Fargate launch type. • Task ENIs add support for jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames will reduce overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC. •	April 8, 2020

Change	Description	Date
	CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see [Monitor Amazon ECS containers using Container Insights with enhanced observability](). • Added support for the task metadata endpoint v4 which provides additional information for your Fargate tasks, including network stats for the task and which Availability Zone the task is running in. For more information, see [Amazon ECS task metadata endpoint version 4](). • Added support for the `SYS_PTRACE` Linux parameter in container definitions. For more information, see [Linux parameters](). • The Fargate container agent replaces the use of the Amazon ECS container agent for all Fargate tasks. This change should not have an effect on how your tasks run. • The container runtime is now using Containerd instead of Docker. This change should not have an effect on how your tasks run. You will notice that some error messages that originate with the container runtime will change from mentioning Docker to more general errors. For more information, see [Fargate platform versions for Amazon ECS]().	

Change	Description	Date
Amazon EFS file system support for task volumes	Amazon EFS file systems can be used as data volumes for both your Amazon ECS and Fargate tasks. For more information, see Use Amazon EFS volumes with Amazon ECS.	April 8, 2020
Amazon ECS Task Metadata Endpoint version 4	Beginning with Amazon ECS container agent version 1.39.0 and Fargate platform version 1.4.0, an environment variable named `ECS_CONTAINER_METADATA_URI_V4` is injected into each container in a task. When you query the task metadata version 4 endpoint, various task metadata and Docker stats are available to tasks. For more information, see Amazon ECS task metadata endpoint version 4.	April 8, 2020
Support for specific versions of Secrets Manager secrets to be injected as environment variables	Added support for specifying sensitive data using specific versions of Secrets Manager secrets. For more information, see Pass sensitive data to an Amazon ECS container.	February 24, 2020
Added additional CodeDeploy deployment configuration options for blue/green deployments	The CodeDeploy service added new canary and linear deployment configurations for the Amazon ECS deployment type. The ability to define custom deployment configurations is also available. For more information, see CodeDeploy blue/green deployments for Amazon ECS.	February 6, 2020
Added the efsVolumeConfiguration task definition parameter	The `efsVolumeConfiguration` task definition parameter is in public preview, which makes it easier to use Amazon EFS file systems with your Amazon ECS tasks. For more information, see Use Amazon EFS volumes with Amazon ECS.	January 17, 2020

Change	Description	Date
Amazon ECS container agent logging behavior updated	The Amazon ECS container agent logging locations and rotation behavior has been updated. For more information, see Amazon ECS container agent log configuration parameters.	January 13, 2020
Fargate Spot	Amazon ECS added support for running tasks using Fargate Spot. For more information, see Amazon ECS clusters for Fargate.	December 3, 2019
Cluster Auto Scaling	Amazon ECS cluster auto scaling enables you to have more control over how you scale tasks within a cluster. For more information, see Automatically manage Amazon ECS capacity with cluster auto scaling.	December 3, 2019
Cluster Capacity Providers	Amazon ECS cluster capacity providers determine the infrastructure to use for your tasks. For more information, see Amazon ECS clusters.	December 3, 2019
Creating a cluster on an AWS Outposts	Amazon ECS now supports creating clusters on an AWS Outposts. For more information, see the section called "Amazon Elastic Container Service on AWS Outposts".	December 3, 2019
Service Action Events	Amazon ECS now sends events to Amazon EventBridge when certain service actions occur. For more information, see Amazon ECS service action events.	November 25, 2019
Amazon ECS GPU-optimized AMI Supports G4 Instances	Amazon ECS added support for the g4 instance type family when using the Amazon ECS GPU-optimized AMI. For more information, see Amazon ECS task definitions for GPU workloads.	October 8, 2019

Change	Description	Date
FireLens for Amazon ECS	FireLens for Amazon ECS is in general availability. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.	September 30, 2019
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Europe (Paris), Europe (Stockholm), and Middle East (Bahrain) regions.	September 30, 2019
Deep Learning Container s with Elastic Inference on Amazon ECS	Amazon ECS supports attaching Amazon Elastic Inference accelerators to your containers to make running deep learning inference workloads more efficient.	September 3, 2019
FireLens for Amazon ECS	FireLens for Amazon ECS is in public preview. FireLens for Amazon ECS enables you to use task definitio n parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Send Amazon ECS logs to an AWS service or AWS Partner.	August 30, 2019
CloudWatch Container Insights	CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Monitor Amazon ECS containers using Container Insights with enhanced observability.	August 30, 2019

Change	Description	Date
Container Level Swap Configuration	Amazon ECS added support for controlling the usage of swap memory space on your Linux container instances at the container level. Using a per-conta iner swap configuration, each container within a task definition can have swap enabled or disabled, and for those that have it enabled, the maximum amount of swap space used can be limited. For more inform ation, see Managing container swap memory space on Amazon ECS.	August 16, 2019
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Hong Kong) Region.	August 6, 2019
Elastic Network Interface Trunking	Added additional supported Amazon EC2 instance types for ENI trunking feature. For more information, see Supported instances for increased Amazon ECS container network interfaces.	August 1, 2019
Registering Multiple Target Groups with a Service	Added support for specifying multiple target groups in a service definition. For more information, see Registering multiple target groups with an Amazon ECS service.	July 30, 2019
Specifying Sensitive Data Using Secrets Manager Secrets	Added tutorial for specifying sensitive data using Secrets Manager secrets. For more information, see Specifying sensitive data using Secrets Manager secrets in Amazon ECS.	July 20, 2019
CloudWatch Container Insights	Amazon ECS has added support for CloudWatc h Container Insights. For more information, see Monitor Amazon ECS containers using Container Insights with enhanced observability.	July 9, 2019

Change	Description	Date
Resource-level permissions for Amazon ECS services and tasksets	Amazon ECS has expanded resource-level permissions support for Amazon ECS services and tasks. For more information, see How Amazon Elastic Container Service works with IAM.	June 27, 2019
New Amazon ECS-optimized AMI patched for AWS-2019-005	Amazon ECS has updated the Amazon ECS-optimized AMI to address the vulnerabilities described in AWS-2019-005.	June 17, 2019
Elastic Network Interface Trunking	Amazon ECS introduces support for launching container instances using supported Amazon EC2 instance types that have increased elastic network interface (ENI) density. Using these instance types and opting in to the `awsvpcTrunking` account setting provides increased ENI density on newly launched container instances which allows you to place more tasks on each container instance. For more information, see Increasing Amazon ECS Linux container instance network interfaces.	June 6, 2019
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports the `splunk` log driver in addition to the `awslogs` log driver. For more information, see Storage and logging.	May 1, 2019
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports referencing sensitive data in the log configuration of a container using the `secretOptions` container definition parameter. For more information, see Pass sensitive data to an Amazon ECS container.	May 1, 2019

Change	Description	Date
AWS Fargate platform version 1.3.0 update	Beginning on April 2, 2019, any new Fargate task that is launched supports injecting sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. For more information, see Pass sensitive data to an Amazon ECS container.	Apr 2, 2019
AWS Fargate platform version 1.3.0 update	Beginning on March 27, 2019, any new Fargate task launched can use additional task definition parameters that enable you to define a proxy configuration, dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Proxy configuration, Container dependency, and Container timeouts.	March 27, 2019
Amazon ECS introduces the external deployment type	The *external* deployment type enables you to use any third-party deployment controller for full control over the deployment process for an Amazon ECS service. For more information, see Deploy Amazon ECS services using a third-party controller.	March 27, 2019
AWS Deep Learning Containers on Amazon ECS	AWS Deep Learning Containers are a set of Docker images for training and serving models in TensorFlow on Amazon Elastic Container Service (Amazon ECS). Deep Learning Containers provide optimized environments with TensorFlow, Nvidia CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries and are available in Amazon ECR. For more information, see Using AWS Deep Learning Containers on Amazon ECS.	March 27, 2019

Change	Description	Date
Amazon ECS introduces enhanced container dependency management	Amazon ECS introduces additional task definition parameters that enable you to define dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see [Container dependency]().	March 7, 2019
Amazon ECS introduces the `PutAccountSettingDefault` API	Amazon ECS introduces the `PutAccountSettingDefault` API that allows a user to set the default ARN/ID format opt in status for all the users and roles on the account. Previously, setting the account's default opt in status required the use of the account owner. For more information, see [Amazon Resource Names (ARNs) and IDs]().	February 8, 2019
Amazon ECS supports GPU workloads	Amazon ECS introduces support for GPU workloads by enabling you to create clusters with GPU-enabled container instances. In a task definition you can specify the number of required GPUs and the ECS agent will pin the physical GPUs to the container. For more information, see [Amazon ECS task definitions for GPU workloads]().	February 4, 2019
Amazon ECS expanded secrets support	Amazon ECS expanded support for using AWS Secrets Manager secrets directly in your task definitions to inject sensitive data into your containers. For more information, see [Pass sensitive data to an Amazon ECS container]().	January 21, 2019

Change	Description	Date
Interface VPC Endpoints (AWS PrivateLink)	Added support for configuring interface VPC endpoints powered by AWS PrivateLink. This allows you to create a private connection between your VPC and Amazon ECS without requiring access over the Internet, through a NAT instance, a VPN connection, or AWS Direct Connect. For more information, see Interface VPC Endpoints (AWS PrivateLink).	December 26, 2018
AWS Fargate platform version 1.3.0	New AWS Fargate platform version released, which contains: • Added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers. For more information, see Pass sensitive data to an Amazon ECS container. • Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service. For more information, see Task retirement and maintenance for AWS Fargate on Amazon ECS. For more information, see Fargate platform versions for Amazon ECS.	December 17, 2018

Change	Description	Date
Service limits updated	The following service limits were updated: • Number of clusters per Region, per account was raised from `1000` to `2000`. • Number of container instances per cluster was raised from `1000` to `2000`. • Number of services per cluster was raised from `500` to `1000`. For more information, see Amazon ECS service quotas.	December 14, 2018
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Mumbai) and Canada (Central) Regions. For more information, see Supported Regions for Amazon ECS on AWS Fargate.	December 7, 2018
Amazon ECS blue/green deployments	Amazon ECS added support for blue/green deployments using CodeDeploy. This deployment type allows you to verify a new deployment of a service before sending production traffic to it. For more information, see CodeDeploy blue/green deployments for Amazon ECS.	November 27, 2018
Amazon ECS-optimized Amazon Linux 2 (arm64) AMI released	Amazon ECS released an Amazon ECS-optimized Amazon Linux 2 AMIs for arm64 architecture. For more information, see Amazon ECS-optimized Linux AMIs.	November 26, 2018

Change	Description	Date
Added support for additional Docker flags in task definitions	Amazon ECS introduced support for the following Docker flags in task definitions: • IPC mode • PID mode	November 16, 2018
Amazon ECS secrets support	Amazon ECS added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers. For more information, see Pass sensitive data to an Amazon ECS container.	November 15, 2018
Resource tagging	Amazon ECS added support for adding metadata tags to your services, task definitions, tasks, clusters, and container instances. For more information, see Tagging Amazon ECS resources.	November 15, 2018
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the US West (N. California) and Asia Pacific (Seoul) Regions. For more information, see AWS Fargate for Amazon ECS.	November 7, 2018

Change	Description	Date
Service limits updated	The following service limits were updated: • Number of tasks using the Fargate launch type, per Region, per account was raised from `20` to `50`. • Number of public IP addresses for tasks using the Fargate launch type was raised from `20` to `50`. For more information, see Amazon ECS service quotas.	October 31, 2018
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Europe (London) Region. For more information, see AWS Fargate for Amazon ECS.	October 26, 2018
Amazon ECS-optimized Amazon Linux 2 AMI Released	Amazon ECS vends Linux AMIs that are optimized for the service in two variants. The latest and recommended version is based on x;. Amazon ECS also vends AMIs that are based on the , but we recommend that you migrate your workloads to the Amazon Linux 2 variant, as support for the Amazon Linux AMI will end no later than June 30, 2020. For more information, see Amazon ECS-optimized Linux AMIs.	October 18, 2018

Change	Description	Date
Amazon ECS Task Metadata Endpoint version 3	Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called `ECS_CONTAINER_METADATA_URI` into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and Docker stats are available to tasks that use the `awsvpc` network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see Monitor workloads using Amazon ECS metadata.	October 18, 2018
Amazon ECS service discovery Region expansion	Amazon ECS service discovery has expanded support to the Canada (Central), South America (São Paulo), Asia Pacific (Seoul), Asia Pacific (Mumbai), and Europe (Paris) Regions. For more information, see Use service discovery to connect Amazon ECS services with DNS names.	September 27, 2018
Added support for additional Docker flags in container definitions	Amazon ECS introduced support for the following Docker flags in container definitions: • System controls • Interactive • Pseudo terminal	September 17, 2018

Change	Description	Date
Private registry authentication support for Amazon ECS using AWS Fargate tasks	Amazon ECS introduced support for Fargate tasks using private registry authentication using AWS Secrets Manager. This feature enables you to store your credentials securely and then reference them in your container definition, which allows your tasks to use private images. For more information, see Using non-AWS container images in Amazon ECS.	September 10, 2018
Amazon ECS service discovery Region expansion	Amazon ECS service discovery has expanded support to the Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), EU (Frankfurt), and Europe (London) Regions. For more information, see Use service discovery to connect Amazon ECS services with DNS names.	August 30, 2018
Scheduled tasks with Fargate tasks support	Amazon ECS introduced support for scheduled tasks for the Fargate launch type.	August 28, 2018
Private registry authentication using AWS Secrets Manager support	Amazon ECS introduced support for private registry authentication using AWS Secrets Manager. This feature enables you to store your credentials securely and then reference them in your container definition, which allows your tasks to use private images. For more information, see Using non-AWS container images in Amazon ECS.	August 16, 2018
Docker volume support added	Amazon ECS introduced support for Docker volumes. For more information, see Storage options for Amazon ECS tasks.	August 9, 2018

Change	Description	Date
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Europe (Frankfurt), Asia Pacific (Singapore), and Asia Pacific (Sydney) Regions. For more information, see AWS Fargate for Amazon ECS.	July 19, 2018
Amazon ECS service scheduler strategies added	Amazon ECS introduced the concept of service scheduler strategies. There are two service scheduler strategies available: • `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica scheduling strategy. • `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constrain ts that you specify in your cluster. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon scheduling strategy. **Note** Fargate tasks do not support the `DAEMON` scheduling strategy. .	June 12, 2018

Change	Description	Date
Amazon ECS container agent v1.18.0	New version of the Amazon ECS container agent released, which added the following functionality: • Added support for customizing the container agent image pull behavior using the `ECS_IMAGE_PULL_BEHAVIOR` parameter. For more information, see Amazon ECS container agent configuration. For more information, see amazon-ecs-agent github.	May 24, 2018
Added Support for `bridge` and `host` Network Modes When Configuring Service Discovery	Added support for configuring service discovery for Amazon ECS services using task definitions that specify the `bridge` or `host` network modes. For more information, see Use service discovery to connect Amazon ECS services with DNS names.	May 22, 2018
Added support for additional Amazon ECS-optimized AMI metadata parameters	Added subparameters that allow you to programatically retrieve the Amazon ECS-optimized AMI ID, image name, operating system, container agent version, and runtime version. Query the metadata using the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-optimized Linux AMI metadata.	May 9, 2018
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the US East (Ohio), US West (Oregon), and EU West (Ireland) Regions. For more information, see AWS Fargate for Amazon ECS.	April 26, 2018

Change	Description	Date
Amazon ECS-optimized AMI Metadata Retrieval	Added ability to programatically retrieve Amazon ECS-optimized AMI metadata using the Systems Manager Parameter Store API. For more informati on, see Retrieving Amazon ECS-optimized Linux AMI metadata.	April 10, 2018
AWS Fargate platform version	New AWS Fargate platform version released, which contains: • Added support for Monitor workloads using Amazon ECS metadata. • Added support for Health check. • Added support for Use service discovery to connect Amazon ECS services with DNS names For more information, see Fargate platform versions for Amazon ECS.	March 26, 2018
Amazon ECS Service Discovery	Added integration with Route 53 to support Amazon ECS service discovery. For more information, see Use service discovery to connect Amazon ECS services with DNS names.	March 22, 2018
Docker shm-size and tmpfs support	Added support for the Docker shm-size and tmpfs parameters in Amazon ECS task definitions. For more information about the updated ECS CLI syntax, see Linux parameters.	March 20, 2018
Container Health Checks	Added support for Docker health checks in container definitions. For more information, see Health check.	March 8, 2018

Change	Description	Date
AWS Fargate	Added overview for Amazon ECS with AWS Fargate. For more information, see AWS Fargate for Amazon ECS.	February 22, 2018
Amazon ECS Task Metadata Endpoint	Beginning with version 1.17.0 of the Amazon ECS container agent, various task metadata and Docker stats are available to tasks that use the `awsvpc` network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see Monitor workloads using Amazon ECS metadata.	February 8, 2018
Amazon ECS Service Auto Scaling using target tracking policies	Added support for ECS Service Auto Scaling using target tracking policies in the Amazon ECS console. For more information, see Use a target metric to scale Amazon ECS services. Removed the previous tutorial for step scaling in the ECS first run wizard. This was replaced with the new tutorial for target tracking.	February 8, 2018
Docker 17.09 support	Added support for Docker 17.09. For more informati on, see Amazon ECS-optimized Linux AMIs.	January 18, 2018
New service scheduler behavior	Updated information about the behavior for service tasks that fail to launch. Documented new service event message that triggers when a service task has consecutive failures. .	January 11, 2018
Elastic Load Balancing health check initializ ation wait period	Added ability to specify a wait period for health checks.	December 27, 2017
Task-level CPU and memory	Added support for specifying CPU and memory at the task-level in task definitions. For more information, see TaskDefinition.	December 12, 2017

Change	Description	Date
Task execution role	The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so it requires an IAM policy and role for the service to know that the agent belongs to you. The following actions are covered by the task execution role: • Calls to Amazon ECR to pull the container image • Calls to CloudWatch to store container application logs For more information, see Amazon ECS task execution IAM role.	December 7, 2017
Windows containers support GA	Added support for Windows Server 2016 container s. For more information, see Amazon ECS-optimized AMI variants.	December 5, 2017
AWS Fargate GA	Added support for launching Amazon ECS services using the Fargate launch type. For more information, see Amazon ECS launch types.	November 29, 2017
Amazon ECS name change	Amazon Elastic Container Service is renamed (previously Amazon EC2 Container Service).	November 21, 2017

Change	Description	Date
Task networking	The task networking features provided by the awsvpc network mode give Amazon ECS tasks the same networking properties as Amazon EC2 instances. When you use the awsvpc network mode in your task definitions, every task that is launched from that task definition gets its own elastic network interface, a primary private IP address, and an internal DNS hostname. The task networking feature simplifies container networking and gives you more control over how containerized applications communicate with each other and other services within your VPCs. For more information, see Amazon ECS task networking options for the EC2 launch type.	November 14, 2017
Amazon ECS container metadata	Amazon ECS containers are now able to access metadata such as their Docker container or image ID, networking configuration, or Amazon ARNs. For more information, see Amazon ECS container metadata file.	November 2, 2017
Docker 17.06 support	Added support for Docker 17.06. For more information, see Amazon ECS-optimized Linux AMIs.	November 2, 2017
Support for Docker flags: device and init	Added support for Docker's device and init features in task definitions using the `LinuxParameters` parameter (`devices` and `initProcessEnabled`). For more information, see LinuxParameters.	November 2, 2017
Support for Docker flags: cap-add and cap-drop	Added support for Docker's cap-add and cap-drop features in task definitions using the `LinuxParameters` parameter (`capabilities`). For more information, see LinuxParameters.	September 22, 2017
Network Load Balancer support	Amazon ECS added support for Network Load Balancers in the Amazon ECS console.	September 7, 2017

Change	Description	Date
RunTask overrides	Added support for task definition overrides when running a task. This allows you to run a task while changing a task definition without the need to create a new task definition revision. For more information, see Running an application as an Amazon ECS task.	June 27, 2017
Amazon ECS scheduled tasks	Added support for scheduling tasks using cron.	June 7, 2017
Spot Instances in the Amazon ECS console	Added support for creating Spot Fleet container instances within the Amazon ECS console. For more information, see Launching an Amazon ECS Linux container instance.	June 6, 2017
Amazon SNS notificat ion for new Amazon ECS-optimized AMI releases	Added ability to subscribe to SNS notifications about new Amazon ECS-optimized AMI releases.	March 23, 2017
Microservices and batch jobs	Added documentation for two common use cases for Amazon ECS: microservices and batch jobs.	February 2017
Container instance draining	Added support for container instance draining, which provides a method for removing container instances from a cluster. For more information, see Draining Amazon ECS container instances.	January 24, 2017
Docker 1.12 support	Added support for Docker 1.12. For more informati on, see Amazon ECS-optimized Linux AMIs.	January 24, 2017
New task placement strategies	Added support for task placement strategies: attribute-based placement, bin pack, Availability Zone spread, and one per host. For more informati on, see Use strategies to define Amazon ECS task placement.	December 29, 2016

Change	Description	Date
Windows container support in beta	Added support for Windows Server 2016 containers (beta). For more information, see Amazon ECS-optimized AMI variants.	December 20, 2016
Blox OSS support	Added support for Blox OSS, which allows for custom task schedulers. For more information, see Schedule your containers on Amazon ECS.	December 1, 2016
Amazon ECS event stream for CloudWatch Events	Amazon ECS now sends container instance and task state changes to CloudWatch Events. For more information, see Automate responses to Amazon ECS errors using EventBridge.	November 21, 2016
Amazon ECS container logging to CloudWatch Logs	Added support for the awslogs driver to send container log streams to CloudWatch Logs. For more information, see Send Amazon ECS logs to CloudWatch .	September 12, 2016
Amazon ECS services with Elastic Load Balancing support for dynamic ports	Added support for a load balancer to support multiple instance:port combinations per listener, which increases flexibility for containers. Now you can let Docker dynamically define the container's host port and the ECS scheduler registers the instance:port with the load balancer. For more information, see Use load balancing to distribute Amazon ECS service traffic.	August 11, 2016
IAM roles for Amazon ECS tasks	Added support for associating IAM roles with a task. This provides finer-grained permissions to containers as opposed to a single role for an entire container instance. For more information, see Amazon ECS task IAM role.	July 13, 2016
Docker 1.11 support	Added support for Docker 1.11. For more information, see Amazon ECS-optimized Linux AMIs.	May 31, 2016

Change	Description	Date
Task automatic scaling	Amazon ECS added support for automatically scaling your tasks run by a service. For more information, see [Automatically scale your Amazon ECS service]().	May 18, 2016
Task definition filtering on task family	Added support for filtering a list of task definitio n based on the task definition family. For more information, see [ListTaskDefinitions]().	May 17, 2016
Docker container and Amazon ECS agent logging	Amazon ECS added ability to send ECS agent and Docker container logs from container instances to CloudWatch Logs to simplify troubleshooting issues.	May 5, 2016
ECS-optimized AMI now supports Amazon Linux 2016.03.	The ECS-optimized AMI added support for Amazon Linux 2016.03. For more information, see [Amazon ECS-optimized Linux AMIs]().	April 5, 2016
Docker 1.9 support	Added support for Docker 1.9. For more information, see [Amazon ECS-optimized Linux AMIs]().	December 22, 2015
CloudWatch metrics for cluster CPU and memory reservation	Amazon ECS added custom CloudWatch metrics for CPU and memory reservation.	December 22, 2015
New Amazon ECS first-run experience	The Amazon ECS console first-run experience added zero-click role creation.	November 23, 2015
Task placement across Availability Zones	The Amazon ECS service scheduler added support for task placement across Availability Zones.	October 8, 2015
CloudWatch metrics for Amazon ECS clusters and services	Amazon ECS added custom CloudWatch metrics for CPU and memory utilization for each container instance, service, and task definition family in a cluster. These new metrics can be used to scale container instances in a cluster using Auto Scaling groups or to create custom CloudWatch alarms.	August 17, 2015

Change	Description	Date
UDP port support	Added support for UDP ports in task definitions.	July 7, 2015
Environment variable overrides	Added support for deregisterTaskDefinition and environment variable overrides for runTask.	June 18, 2015
Automated Amazon ECS agent updates	Added ability to see the ECS agent version that is running on a container instance. Also able to update the ECS agent from the AWS Management Console, AWS CLI, and SDK.	June 11, 2015
Amazon ECS service scheduler and Elastic Load Balancing integrati on	Added ability to define a service and associate that service with an Elastic Load Balancing load balancer.	April 9, 2015
Amazon ECS GA	Amazon ECS general availability in the US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), and Europe (Ireland) Regions.	April 9, 2015