# Table of Contents

What is Amazon ECS ................................................................. 1
  Features of Amazon ECS ........................................................ 1
    Containers and Images ......................................................... 2
    Task Definitions ................................................................. 3
    Tasks and Scheduling .......................................................... 4
    Clusters .............................................................................. 4
  Container Agent ................................................................. 4
  How to Get Started with Amazon ECS ................................. 5
Related Services ................................................................. 5
Accessing Amazon ECS ....................................................... 6

Setting Up ................................................................................ 7
  Sign Up for AWS ............................................................... 7
  Create an IAM User ............................................................ 7
  Create an IAM Role ............................................................ 9
  Create a Key Pair ............................................................... 9
  Create a Virtual Private Cloud ............................................ 11
  Create a Security Group ..................................................... 12
  Install the AWS CLI ............................................................ 13

Docker Basics for Amazon ECS .............................................. 14
  Installing Docker ............................................................... 14
  Create a Docker Image ....................................................... 15
  (Optional) Push your image to Amazon Elastic Container Registry .............................................. 16
  (Optional) Clean up ............................................................ 18
  Next Steps .......................................................................... 18

Getting Started with Amazon ECS ........................................... 19
  Getting Started with Amazon ECS using Fargate .................... 19
    Prerequisites .................................................................... 19
    Step 1: Create a Task Definition ....................................... 20
    Step 2: Configure the Service .......................................... 20
    Step 3: Configure the Cluster .......................................... 21
    Step 4: Review ............................................................... 21
    Step 5: (Optional) View your Service .................................. 21
  Getting Started with Amazon ECS ........................................ 22
    Prerequisites .................................................................... 22
    Step 1: Choose Your Configuration Options ....................... 23
    Step 2: Create a Task Definition ....................................... 24
    Step 3: Configure the Service .......................................... 25
    Step 4: Configure the Cluster .......................................... 25
    Step 5: Review ............................................................... 26
    Step 6: (Optional) View your Service .................................. 26

Cleaning Up ............................................................................ 27
  Scale Down Services .......................................................... 27
  Delete Services ................................................................. 27
  Deregister Container Instances .......................................... 27
  Delete a Cluster ................................................................. 28
  Delete the AWS CloudFormation Stack ................................. 28

AWS Fargate ........................................................................... 30
  Task Definitions ................................................................... 30
    Network Mode ................................................................. 31
    Task CPU and Memory .................................................... 31
    Logging ............................................................................ 32
    Amazon ECS Task Execution IAM Role .............................. 32
  Example Task Definition ................................................... 32
  Task Storage ....................................................................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Version 2014-11-13</td>
<td>iv</td>
</tr>
<tr>
<td>Amazon Elastic Container Service Developer Guide</td>
<td></td>
</tr>
<tr>
<td>Tasks and Services</td>
<td>34</td>
</tr>
<tr>
<td>Task Networking</td>
<td>34</td>
</tr>
<tr>
<td>Private Registry Authentication</td>
<td>34</td>
</tr>
<tr>
<td>Clusters</td>
<td>35</td>
</tr>
<tr>
<td>Fargate Task Retirement</td>
<td>33</td>
</tr>
<tr>
<td>AWS Fargate Platform Versions</td>
<td>36</td>
</tr>
<tr>
<td>Platform Version Considerations</td>
<td>36</td>
</tr>
<tr>
<td>Available Platform Versions</td>
<td>36</td>
</tr>
<tr>
<td>Working with GPUs on Amazon ECS</td>
<td>38</td>
</tr>
<tr>
<td>Considerations for Working with GPUs</td>
<td>38</td>
</tr>
<tr>
<td>Specifying GPUs in Your Task Definition</td>
<td>39</td>
</tr>
<tr>
<td>Tutorials</td>
<td>41</td>
</tr>
<tr>
<td>Tutorial: Creating a Service Using a Blue/Green Deployment</td>
<td>41</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>41</td>
</tr>
<tr>
<td>Step 1: Create an Application Load Balancer</td>
<td>41</td>
</tr>
<tr>
<td>Step 2: Create an Amazon ECS Cluster</td>
<td>42</td>
</tr>
<tr>
<td>Step 3: Register a Task Definition</td>
<td>42</td>
</tr>
<tr>
<td>Step 4: Create an Amazon ECS Service</td>
<td>43</td>
</tr>
<tr>
<td>Step 5: Create the AWS CodeDeploy Resources</td>
<td>44</td>
</tr>
<tr>
<td>Step 5: Create and Monitor an CodeDeploy Deployment</td>
<td>46</td>
</tr>
<tr>
<td>Step 6: Clean Up</td>
<td>48</td>
</tr>
<tr>
<td>Tutorial: Continuous Deployment with CodePipeline</td>
<td>49</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>49</td>
</tr>
<tr>
<td>Step 1: Add a Build Specification File to Your Source Repository</td>
<td>50</td>
</tr>
<tr>
<td>Step 2: Creating Your Continuous Deployment Pipeline</td>
<td>51</td>
</tr>
<tr>
<td>Step 3: Add Amazon ECR Permissions to the CodeBuild Role</td>
<td>52</td>
</tr>
<tr>
<td>Step 4: Test Your Pipeline</td>
<td>53</td>
</tr>
<tr>
<td>Clusters</td>
<td>55</td>
</tr>
<tr>
<td>Creating a Cluster</td>
<td>55</td>
</tr>
<tr>
<td>Scaling a Cluster</td>
<td>57</td>
</tr>
<tr>
<td>Deleting a Cluster</td>
<td>59</td>
</tr>
<tr>
<td>Container Instances</td>
<td>60</td>
</tr>
<tr>
<td>Container Instance Concepts</td>
<td>60</td>
</tr>
<tr>
<td>Container Instance Lifecycle</td>
<td>61</td>
</tr>
<tr>
<td>Check the Instance Role for Your Account</td>
<td>61</td>
</tr>
<tr>
<td>Amazon ECS-optimized AMIs</td>
<td>62</td>
</tr>
<tr>
<td>How to Launch the Latest Amazon ECS-Optimized Amazon Linux AMI</td>
<td>72</td>
</tr>
<tr>
<td>AMI Versions</td>
<td>82</td>
</tr>
<tr>
<td>AMI Storage Configuration</td>
<td>87</td>
</tr>
<tr>
<td>Retrieving Amazon ECS-Optimized AMI Metadata</td>
<td>91</td>
</tr>
<tr>
<td>Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications</td>
<td>97</td>
</tr>
<tr>
<td>Amazon SNS Message Format</td>
<td>99</td>
</tr>
<tr>
<td>Launching a Container Instance</td>
<td>101</td>
</tr>
<tr>
<td>Bootstrap Container Instances</td>
<td>105</td>
</tr>
<tr>
<td>Amazon ECS Container Agent</td>
<td>106</td>
</tr>
<tr>
<td>Docker Daemon</td>
<td>106</td>
</tr>
<tr>
<td>cloud-init-per Utility</td>
<td>107</td>
</tr>
<tr>
<td>Specifying Multiple User Data Blocks Using a MIME Multi Part Archive</td>
<td>107</td>
</tr>
<tr>
<td>Example User Data Scripts</td>
<td>108</td>
</tr>
<tr>
<td>Connect to Your Container Instance</td>
<td>113</td>
</tr>
<tr>
<td>CloudWatch Logs</td>
<td>114</td>
</tr>
<tr>
<td>CloudWatch Logs IAM Policy</td>
<td>114</td>
</tr>
<tr>
<td>Installing the CloudWatch Logs Agent</td>
<td>115</td>
</tr>
<tr>
<td>Configuring and Starting the CloudWatch Logs Agent</td>
<td>115</td>
</tr>
<tr>
<td>Viewing CloudWatch Logs</td>
<td>118</td>
</tr>
<tr>
<td>Configuring CloudWatch Logs at Launch with User Data</td>
<td>119</td>
</tr>
<tr>
<td>Container Instance Draining</td>
<td>122</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>552</td>
</tr>
<tr>
<td>Step 1: (Optional) Create a Cluster</td>
<td>553</td>
</tr>
<tr>
<td>Step 2: Register a Task Definition</td>
<td>553</td>
</tr>
<tr>
<td>Step 3: List Task Definitions</td>
<td>555</td>
</tr>
<tr>
<td>Step 4: Create a Service</td>
<td>555</td>
</tr>
<tr>
<td>Step 5: List Services</td>
<td>556</td>
</tr>
<tr>
<td>Step 6: Describe the Running Service</td>
<td>557</td>
</tr>
<tr>
<td>Tutorial: Creating a Cluster with an EC2 Task Using the AWS CLI</td>
<td>558</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>559</td>
</tr>
<tr>
<td>Step 1: (Optional) Create a Cluster</td>
<td>559</td>
</tr>
<tr>
<td>Step 2: Launch an Instance with the Amazon ECS AMI</td>
<td>559</td>
</tr>
<tr>
<td>Step 3: List Container Instances</td>
<td>561</td>
</tr>
<tr>
<td>Step 4: Describe your Container Instance</td>
<td>561</td>
</tr>
<tr>
<td>Step 5: Register a Task Definition</td>
<td>563</td>
</tr>
<tr>
<td>Step 6: List Task Definitions</td>
<td>564</td>
</tr>
<tr>
<td>Step 7: Run a Task</td>
<td>565</td>
</tr>
<tr>
<td>Step 8: List Tasks</td>
<td>565</td>
</tr>
<tr>
<td>Step 9: Describe the Running Task</td>
<td>566</td>
</tr>
<tr>
<td>Common Use Cases</td>
<td>567</td>
</tr>
<tr>
<td>Microservices</td>
<td>567</td>
</tr>
<tr>
<td>Auto Scaling</td>
<td>567</td>
</tr>
<tr>
<td>Service Discovery</td>
<td>568</td>
</tr>
<tr>
<td>Authorization and Secrets Management</td>
<td>568</td>
</tr>
<tr>
<td>Logging</td>
<td>568</td>
</tr>
<tr>
<td>Continuous Integration and Continuous Deployment</td>
<td>568</td>
</tr>
<tr>
<td>Batch Jobs</td>
<td>569</td>
</tr>
<tr>
<td>App Mesh and Amazon ECS</td>
<td>570</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>570</td>
</tr>
<tr>
<td>Step 1: Create Your Service Mesh</td>
<td>570</td>
</tr>
<tr>
<td>Step 2: Create Your Virtual Nodes</td>
<td>570</td>
</tr>
<tr>
<td>Step 3: Create Your Virtual Routers</td>
<td>571</td>
</tr>
<tr>
<td>Step 4: Create Your Routes</td>
<td>572</td>
</tr>
<tr>
<td>Step 5: Create Your Virtual Services</td>
<td>572</td>
</tr>
<tr>
<td>Update Your Microservice Task Definitions</td>
<td>573</td>
</tr>
<tr>
<td>Proxy Configuration</td>
<td>573</td>
</tr>
<tr>
<td>Application Container Envoy Dependency</td>
<td>574</td>
</tr>
<tr>
<td>Envoy Container Definition</td>
<td>574</td>
</tr>
<tr>
<td>Example Task Definition</td>
<td>575</td>
</tr>
<tr>
<td>AWS Deep Learning Containers on Amazon ECS</td>
<td>577</td>
</tr>
<tr>
<td>Tutorial: Creating a VPC</td>
<td>578</td>
</tr>
<tr>
<td>Step 1: Create an Elastic IP Address for Your NAT Gateway</td>
<td>578</td>
</tr>
<tr>
<td>Step 2: Run the VPC Wizard</td>
<td>578</td>
</tr>
<tr>
<td>Step 3: Create Additional Subnets</td>
<td>579</td>
</tr>
<tr>
<td>Next Steps</td>
<td>579</td>
</tr>
<tr>
<td>Tutorial: Using Amazon EFS</td>
<td>581</td>
</tr>
<tr>
<td>Step 1: Gather Cluster Information</td>
<td>581</td>
</tr>
<tr>
<td>Step 2: Create a Security Group for an Amazon EFS File System</td>
<td>582</td>
</tr>
<tr>
<td>Step 3: Create an Amazon EFS File System</td>
<td>582</td>
</tr>
<tr>
<td>Step 4: Configure Container Instances</td>
<td>583</td>
</tr>
<tr>
<td>Step 5: Create a Task Definition to Use the Amazon EFS File System</td>
<td>584</td>
</tr>
<tr>
<td>Step 6: Add Content to the Amazon EFS File System</td>
<td>585</td>
</tr>
<tr>
<td>Step 7: Run a Task and View the Results</td>
<td>586</td>
</tr>
<tr>
<td>Service Limits</td>
<td>587</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>589</td>
</tr>
<tr>
<td>Troubleshooting First-Run Wizard Launch Issues</td>
<td>589</td>
</tr>
<tr>
<td>Checking Stopped Tasks for Errors</td>
<td>590</td>
</tr>
<tr>
<td>Service Event Messages</td>
<td>591</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Service Event Messages</td>
<td>592</td>
</tr>
<tr>
<td>Invalid CPU or Memory Value Specified</td>
<td>594</td>
</tr>
<tr>
<td>Cannot Pull Container Image Error</td>
<td>595</td>
</tr>
<tr>
<td>CannotCreateContainerError: API error (500): devmapper</td>
<td>596</td>
</tr>
<tr>
<td>Troubleshooting Service Load Balancers</td>
<td>597</td>
</tr>
<tr>
<td>Enabling Docker Debug Output</td>
<td>599</td>
</tr>
<tr>
<td>Amazon ECS Log File Locations</td>
<td>600</td>
</tr>
<tr>
<td>Amazon ECS Container Agent Log</td>
<td>600</td>
</tr>
<tr>
<td>Amazon ECS ecs-init Log</td>
<td>600</td>
</tr>
<tr>
<td>IAM Roles for Tasks Credential Audit Log</td>
<td>601</td>
</tr>
<tr>
<td>Troubleshooting IAM Roles for Tasks</td>
<td>607</td>
</tr>
<tr>
<td>Amazon ECS Logs Collector</td>
<td>601</td>
</tr>
<tr>
<td>Agent Introspection Diagnostics</td>
<td>602</td>
</tr>
<tr>
<td>Docker Diagnostics</td>
<td>604</td>
</tr>
<tr>
<td>List Docker Containers</td>
<td>604</td>
</tr>
<tr>
<td>View Docker Logs</td>
<td>605</td>
</tr>
<tr>
<td>Inspect Docker Containers</td>
<td>605</td>
</tr>
<tr>
<td>API failures Error Messages</td>
<td>606</td>
</tr>
<tr>
<td>Amazon ECS Logs Collector</td>
<td>601</td>
</tr>
<tr>
<td>Agent Introspection Diagnostics</td>
<td>602</td>
</tr>
<tr>
<td>Docker Diagnostics</td>
<td>604</td>
</tr>
<tr>
<td>List Docker Containers</td>
<td>604</td>
</tr>
<tr>
<td>View Docker Logs</td>
<td>605</td>
</tr>
<tr>
<td>Inspect Docker Containers</td>
<td>605</td>
</tr>
<tr>
<td>Windows Containers</td>
<td>610</td>
</tr>
<tr>
<td>Getting Started with Windows Containers</td>
<td>611</td>
</tr>
<tr>
<td>Step 1: Create a Windows Cluster</td>
<td>611</td>
</tr>
<tr>
<td>Step 2: Launching a Windows Container Instance into your Cluster</td>
<td>612</td>
</tr>
<tr>
<td>Step 3: Register a Windows Task Definition</td>
<td>614</td>
</tr>
<tr>
<td>Step 4: Create a Service with Your Task Definition</td>
<td>615</td>
</tr>
<tr>
<td>Step 5: View Your Service</td>
<td>616</td>
</tr>
<tr>
<td>Windows Task Definitions</td>
<td>616</td>
</tr>
<tr>
<td>Windows Task Definition Parameters</td>
<td>616</td>
</tr>
<tr>
<td>Windows Sample Task Definitions</td>
<td>618</td>
</tr>
<tr>
<td>Windows IAM Roles for Tasks</td>
<td>619</td>
</tr>
<tr>
<td>IAM Roles for Task Container Bootstrap Script</td>
<td>620</td>
</tr>
<tr>
<td>Pushing Windows Images to Amazon ECR</td>
<td>620</td>
</tr>
<tr>
<td>Document History</td>
<td>622</td>
</tr>
<tr>
<td>AWS Glossary</td>
<td>637</td>
</tr>
</tbody>
</table>
What is Amazon Elastic Container Service?

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster. You can host your cluster on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks using the Fargate launch type. For more control you can host your tasks on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances that you manage by using the EC2 launch type. For more information about launch types, see Amazon ECS Launch Types (p. 227).

Amazon ECS lets you launch and stop container-based applications with simple API calls, allows you to get the state of your cluster from a centralized service, and gives you access to many familiar Amazon EC2 features.

You can use Amazon ECS to schedule the placement of containers across your cluster based on your resource needs, isolation policies, and availability requirements. Amazon ECS eliminates the need for you to operate your own cluster management and configuration management systems or worry about scaling your management infrastructure.

Amazon ECS can be used to create a consistent deployment and build experience, manage, and scale batch and Extract-Transform-Load (ETL) workloads, and build sophisticated application architectures on a microservices model. For more information about Amazon ECS use cases and scenarios, see Container Use Cases.

AWS Elastic Beanstalk can also be used to rapidly develop, test, and deploy Docker containers in conjunction with other components of your application infrastructure; however, using Amazon ECS directly provides more fine-grained control and access to a wider set of use cases. For more information, see the AWS Elastic Beanstalk Developer Guide.

Features of Amazon ECS

Amazon ECS is a regional service that simplifies running application containers in a highly available manner across multiple Availability Zones within a Region. You can create Amazon ECS clusters within a new or existing VPC. After a cluster is up and running, you can define task definitions and services that specify which Docker container images to run across your clusters. Container images are stored in and pulled from container registries, which may exist within or outside of your AWS infrastructure.

The following diagram shows the architecture of an Amazon ECS environment using the Fargate launch type:
The following sections dive into these individual elements of the Amazon ECS architecture in more detail.

## Containers and Images

To deploy applications on Amazon ECS, your application components must be architected to run in containers. A Docker container is a standardized unit of software development, containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc. Containers are created from a read-only template called an image.
Images are typically built from a Dockerfile, a plain text file that specifies all of the components that are included in the container. These images are then stored in a registry from which they can be downloaded and run on your cluster. For more information about container technology, see Docker Basics for Amazon ECS (p. 14).

Task Definitions

To prepare your application to run on Amazon ECS, you create a task definition. The task definition is a text file, in JSON format, that describes one or more containers, up to a maximum of ten, that form your application. It can be thought of as a blueprint for your application. Task definitions specify various parameters for your application. Examples of task definition parameters are which containers to use, which launch type to use, which ports should be opened for your application, and what data volumes should be used with the containers in the task. The specific parameters available for the task definition depend on which launch type you are using. For more information about creating task definitions, see Amazon ECS Task Definitions (p. 186).

The following is an example of a task definition containing a single container that runs an NGINX web server using the Fargate launch type. For a more extended example demonstrating the use of multiple containers in a task definition, see Example Task Definitions (p. 260).

```json
{
    "family": "webserver",
    "containerDefinitions": [
        {
            "name": "web",
            "image": "nginx",
            "memory": "100",
            "cpu": "99"
        },
        ...
    ],
    "requiresCompatibilities": ["FARGATE"],
    "networkMode": "awsvpc",
    "memory": "512",
    "cpu": "256",
}
```
Tasks and Scheduling

A task is the instantiation of a task definition within a cluster. After you have created a task definition for your application within Amazon ECS, you can specify the number of tasks that will run on your cluster.

Each task that uses the Fargate launch type has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

The Amazon ECS task scheduler is responsible for placing tasks within your cluster. There are several different scheduling options available. For example, you can define a service that runs and maintains a specified number of tasks simultaneously. For more information about the different scheduling options available, see Scheduling Amazon ECS Tasks (p. 268).

Clusters

When you run tasks using Amazon ECS, you place them on a cluster, which is a logical grouping of resources. When using the Fargate launch type with tasks within your cluster, Amazon ECS manages your cluster resources. When using the EC2 launch type, then your clusters are a group of container instances you manage. An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent. Amazon ECS downloads your container images from a registry that you specify, and runs those images within your cluster.

For more information about creating clusters, see Amazon ECS Clusters (p. 55). If you are using the EC2 launch type, you can read about creating container instances at Amazon ECS Container Instances (p. 60).

Container Agent

The container agent runs on each infrastructure resource within an Amazon ECS cluster. It sends information about the resource's current running tasks and resource utilization to Amazon ECS, and starts and stops tasks whenever it receives a request from Amazon ECS. For more information, see Amazon ECS Container Agent (p. 133).
How to Get Started with Amazon ECS

If you are using Amazon ECS for the first time, the AWS Management Console for Amazon ECS provides a first-run wizard that steps you through defining a task definition for a web server, configuring a service, and launching your first Fargate task. The first-run wizard is highly recommended for users who have no prior experience with Amazon ECS. For more information, see the Getting Started with Amazon ECS (p. 19) tutorial.

Alternatively, you can install the AWS Command Line Interface (AWS CLI) to use Amazon ECS. For more information, see Setting Up with Amazon ECS (p. 7).

Related Services

Amazon ECS can be used along with the following AWS services:

**AWS Identity and Access Management**

IAM is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization). In Amazon ECS, IAM can be used to control access at the container instance level using IAM roles, and at the task level using IAM task roles. For more information, see Amazon ECS IAM Policies, Roles, and Permissions (p. 398).

**Amazon EC2 Auto Scaling**

Auto Scaling is a web service that enables you to automatically scale out or in your tasks based on user-defined policies, health status checks, and schedules. You can use Auto Scaling with a Fargate
task within a service to scale in response to a number of metrics or with an EC2 task to scale the container instances within your cluster. For more information, see Service Auto Scaling (p. 321).

**Elastic Load Balancing**

Elastic Load Balancing automatically distributes incoming application traffic across the tasks in your Amazon ECS service. It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic. You can use Elastic Load Balancing to create an endpoint that balances traffic across services in a cluster. For more information, see Service Load Balancing (p. 306).

**Amazon Elastic Container Registry**

Amazon ECR is a managed AWS Docker registry service that is secure, scalable, and reliable. Amazon ECR supports private Docker repositories with resource-based permissions using IAM so that specific users or tasks can access repositories and images. Developers can use the Docker CLI to push, pull, and manage images. For more information, see the Amazon Elastic Container Registry User Guide.

**AWS CloudFormation**

AWS CloudFormation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable fashion. You can define clusters, task definitions, and services as entities in an AWS CloudFormation script. For more information, see AWS CloudFormation Template Reference.

### Accessing Amazon ECS

You can work with Amazon ECS in the following ways:

**AWS Management Console**

The console is a browser-based interface to manage Amazon ECS resources. For a tutorial that guides you through the console, see Getting Started with Amazon ECS (p. 19).

**AWS command line tools**

You can use the AWS command line tools to issue commands at your system's command line to perform Amazon ECS and AWS tasks; this can be faster and more convenient than using the console. The command line tools are also useful for building scripts that perform AWS tasks.

AWS provides two sets of command line tools: the AWS Command Line Interface (AWS CLI) and the AWS Tools for Windows PowerShell. For more information, see the AWS Command Line Interface User Guide and the AWS Tools for Windows PowerShell User Guide.

**Amazon ECS CLI**

In addition to using the AWS CLI to access Amazon ECS resources, you can use the Amazon ECS CLI, which provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment using Docker Compose. For more information, see Using the Amazon ECS Command Line Interface (p. 450).

**AWS SDKs**

We also provide SDKs that enable you to access Amazon ECS from a variety of programming languages. The SDKs automatically take care of tasks such as:

- Cryptographically signing your service requests
- Retrying requests
- Handling error responses

For more information about available SDKs, see Tools for Amazon Web Services.
Setting Up with 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 cluster using either the Amazon ECS first-run wizard or the Amazon ECS Command Line Interface (CLI).

**Note**
Because Amazon ECS uses many components of Amazon EC2, you use the Amazon EC2 console for many of these steps.

Complete the following tasks to get set up for Amazon ECS. If you have already completed any of these steps, you may skip them and move on to installing the custom AWS CLI.

1. **Sign Up for AWS** (p. 7)
2. **Create an IAM User** (p. 7)
3. **Create an IAM Role** (p. 9)
4. **Create a Key Pair** (p. 9)
5. **Create a Virtual Private Cloud** (p. 11)
6. **Create a Security Group** (p. 12)
7. **Install the AWS CLI** (p. 13)

**Sign Up for AWS**

When you sign up for AWS, your AWS account is automatically signed up for all services, including Amazon EC2 and Amazon ECS. You are charged only for the services that you use.

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

**To create an AWS account**

1. Open [https://aws.amazon.com/](https://aws.amazon.com/), and then choose **Create an AWS Account**.

   **Note**
   If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code using the phone keypad.

   **Note** your AWS account number, because you'll need it for the next task.

**Create an IAM User**

Services in AWS, such as Amazon EC2 and Amazon ECS, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The
console requires your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console.

To create an IAM user for yourself and add the user to an Administrators group

1. Use your AWS account email address and password to sign in as the AWS account root user to the IAM console at https://console.aws.amazon.com/iam/.

   **Note**
   We strongly recommend that you adhere to the best practice of using the Administrator IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.

2. In the navigation pane of the console, choose Users, and then choose Add user.

3. For User name, type Administrator.

4. Select the check box next to AWS Management Console access, select Custom password, and then type the new user's password in the text box. You can optionally select Require password reset to force the user to create a new password the next time the user signs in.

5. Choose Next: Permissions.

6. On the Set permissions page, choose Add user to group.

7. Choose Create group.

8. In the Create group dialog box, for Group name type Administrators.

9. For Filter policies, select the check box for AWS managed - job function.

10. In the policy list, select the check box for AdministratorAccess. Then choose Create group.

11. Back in the list of groups, select the check box for your new group. Choose Refresh if necessary to see the group in the list.

12. Choose Next: Tags to add metadata to the user by attaching tags as key-value pairs.

13. Choose Next: Review to see the list of group memberships to be added to the new user. When you are ready to proceed, choose Create user.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to Access Management and Example Policies.

To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where your_aws_account_id is your AWS account number without the hyphens (for example, if your AWS account number is 1234-5678-9012, your AWS account ID is 123456789012):

https://your_aws_account_id.signin.aws.amazon.com/console/

Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "your_user_name @ your_aws_account_id".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. From the IAM dashboard, choose Create Account Alias and enter an alias, such as your company name. To sign in after you create an account alias, use the following URL:

https://your_account_alias.signin.aws.amazon.com/console/
Create an IAM Role

Before the Amazon ECS container agent can make calls to the Amazon ECS API actions on your behalf, it requires an IAM policy and role for the service to know that the agent belongs to you.

For tasks using the EC2 launch type, you can create an IAM role that allows the agent to know which account it should register your container instances with. When you launch a container instance with the Amazon ECS-optimized AMI provided by Amazon using this role, the agent automatically registers the container instance into your default cluster. This role is referred to as the Amazon ECS container instance IAM role. For more information, see Amazon ECS Container Instance IAM Role (p. 415).

The Amazon ECS container agent also makes calls to the Amazon EC2 and Elastic Load Balancing APIs on your behalf; so container instances can be registered and deregistered with load balancers. Before you can attach a load balancer to an Amazon ECS service, you must create an IAM role for your services to use before you start them. This requirement applies to any Amazon ECS service that you plan to use with a load balancer. This role is referred to as the Amazon ECS service scheduler IAM role. For more information, see Amazon ECS Service Scheduler IAM Role (p. 426).

For tasks using the Fargate launch type, you can create an IAM role that allows the agent to pull container images from Amazon ECR or to use the awslogs log driver, which is currently the only supported logging option for this launch type. This role is referred to as the Amazon ECS task execution IAM role. For more information, see Amazon ECS Task Execution IAM Role (p. 418).

Note
These IAM roles are automatically created for you in the Amazon ECS console first-run experience, so if you intend to use the console, you can move ahead to the next section. If you do not intend to use the console, and instead plan to use the AWS CLI, these IAM roles will need to be manually created.

Create a Key Pair

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 for Linux Instances.

To create a key pair
1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a Region for the key pair. You can select any Region that’s available to you, regardless of your location. However, key pairs are specific to a Region. For example, if you plan to launch a container instance in the US East (Ohio) Region, you must create a key pair for the instance in the US East (Ohio) Region.
3. In the navigation pane, under **NETWORK & SECURITY**, choose **Key Pairs**.

   **Tip**
   The navigation pane is on the left side of the console. If you do not see the pane, it might be minimized; choose the arrow to expand the pane. You may have to scroll down to see the **Key Pairs** link.

4. Choose **Create Key Pair**.

5. Enter a name for the new key pair in the **Key pair name** field of the **Create Key Pair** dialog box, and then choose **Create**. Use a name that is easy for you to remember, such as your IAM user name, followed by `-key-pair`, plus the region name. For example, `me-key-pair-useast2`. 
6. The private key file is automatically downloaded by your browser. The base file name is the name you specified as the name of your key pair, and the file name extension is .pem. Save the private key file in a safe place.

**Important**
This is the only chance for you to save the private key file. Provide the name of your key pair when you launch an instance and the corresponding private key each time you connect to the instance.

7. If you use an SSH client on a macOS or Linux computer to connect to your Linux instance, use the following command to set the permissions of your private key file so that only you can read it.

```
chmod 400 your_user_name-key-pair-region_name.pem
```

For more information, see Amazon EC2 Key Pairs in the Amazon EC2 User Guide for Linux Instances.

**To connect to your instance using your key pair**

To connect to your Linux instance from a computer running macOS or Linux, specify the .pem file to your SSH client with the `-i` option and the path to your private key. To connect to your Linux instance from a computer running Windows, you can use either MindTerm or PuTTY. If you plan to use PuTTY, you need to install it and use the following procedure to convert the .pem file to a .ppk file.

(Optional) To prepare to connect to a Linux instance from Windows using PuTTY

1. Download and install PuTTY from http://www.chiark.greenend.org.uk/~sgtatham/putty/. Be sure to install the entire suite.
2. Start PuTTYgen (for example, from the Start menu, choose All Programs > PuTTY > PuTTYgen).
3. Under Type of key to generate, choose RSA.
4. Choose Load. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, select the option to display files of all types.
5. Select the private key file that you created in the previous procedure and choose Open. Choose OK to dismiss the confirmation dialog box.
6. Choose Save private key. PuTTYgen displays a warning about saving the key without a passphrase. Choose Yes.
7. Specify the same name for the key that you used for the key pair. PuTTY automatically adds the .ppk file extension.

---

Create a Virtual Private Cloud

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

Note
The Amazon ECS console first run experience creates a security group for your instances and load balancer based on the task definition you use, so if you intend to use the Amazon ECS console, you can move ahead to the next section.

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 for Linux Instances.

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.

To create a security group with least privilege

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a Region for the security group. Security groups are specific to a Region, so you should select the same Region in which you created your key pair.


4. Enter a name for the new security group and a description. Choose a name that is easy for you to remember, such as ecs-instances-default-cluster.

5. In the VPC list, ensure that your default VPC is selected. It’s marked with an asterisk (*).

   **Note**
   If your account supports Amazon EC2 Classic, select the VPC that you created in the previous task.

6. 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.

   On the **Inbound** tab, create the following rules (choose **Add Rule** for each new rule), and then choose **Create**:

   - Choose HTTP from the **Type** list, and make sure that **Source** is set to **Anywhere** (0.0.0.0/0).
   - Choose HTTPS from the **Type** list, and make sure that **Source** is set to **Anywhere** (0.0.0.0/0).
   - Choose SSH from the **Type** list. In the **Source** field, ensure that **Custom IP** is selected, and 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.

---

### Install the AWS CLI

The AWS Management Console can be used to manage all operations manually with Amazon ECS. However, installing the AWS CLI on your local desktop or a developer box enables you to 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 the AWS Command Line Interface in the AWS Command Line Interface User Guide.
Docker Basics for Amazon ECS

Docker is a technology that allows you to build, run, test, and deploy distributed applications that are based on Linux containers. Amazon ECS uses Docker images in task definitions to launch containers on Amazon EC2 instances in your clusters. For Amazon ECS product details, featured customer case studies, and FAQs, see the Amazon Elastic Container Service product detail pages.

The documentation in this guide assumes that readers possess a basic understanding of what Docker is and how it works. For more information about Docker, see What is Docker? and the Docker overview.

Topics
• Installing Docker (p. 14)
• Create a Docker Image (p. 15)
• (Optional) Push your image to Amazon Elastic Container Registry (p. 16)
• (Optional) Clean up (p. 18)
• Next Steps (p. 18)

Installing Docker

Note
If you already have Docker installed, skip to Create a Docker Image (p. 15).

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

You don't even need a local development system to use Docker. If you are using Amazon EC2 already, you can launch an instance and install Docker to get started.

To install Docker on an Amazon EC2 instance

1. Launch an instance with the Amazon Linux 2 AMI. For more information, see Launching an Instance in the Amazon EC2 User Guide for Linux Instances.
2. Connect to your instance. For more information, see Connect to Your Linux Instance in the Amazon EC2 User Guide for Linux Instances.
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 amazon-linux-extras 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.
Create a Docker Image

Amazon ECS task definitions use Docker 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 EC2 instance, and then push the image to a container registry (such as Amazon ECR or Docker Hub) so you can use it in an 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.

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

```plaintext
FROM ubuntu:16.04

# Install dependencies
RUN apt-get update
RUN apt-get -y install apache2

# Install apache and write hello world message
RUN echo 'Hello World!' > /var/www/html/index.html

# Configure apache
RUN echo './etc/apache2/envvars' > /root/run_apache.sh
RUN echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh
RUN echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh
RUN echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh
RUN chmod 755 /root/run_apache.sh

EXPOSE 80

CMD /root/run_apache.sh
```

This Dockerfile uses the Ubuntu 16.04 image. The RUN instructions update the package caches, install some software packages for the web server, and then write the "Hello World!" content to the

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.

```plaintext
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?
```
web server’s document root. The EXPOSE instruction exposes port 80 on the container, 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.

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

4. Run `docker images` to verify that the image was created correctly.

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

    Output:

    | REPOSITORY   | TAG     | IMAGE ID       | CREATED            | SIZE   |
    |--------------|---------|----------------|---------------------|--------|
    | hello-world  | latest  | e9ffedc8c286   | 4 minutes ago       | 241MB  |

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. For more information about `docker run`, go to the Docker run reference.

    ```
    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 server’s 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/.
   - If you are using `docker-machine` on a Windows or Mac computer, find the IP address of the VirtualBox VM that is hosting Docker with the `docker-machine ip` command, substituting `machine-name` with the name of the docker machine you are using.

    ```
    docker-machine ip machine-name
    ```

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

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

(Optional) Push your image to Amazon Elastic Container Registry

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

This section requires the following:
You have the AWS CLI installed and configured. If you do not have the AWS CLI installed on your system, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.

Your user has the required IAM permissions to access the Amazon ECR service. For more information, see Amazon ECR Managed Policies.

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.

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

   Output:

   ```json
   {
     "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.

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

3. Run the `aws ecr get-login --no-include-email` command to get the `docker login` authentication command string for your registry.

   **Note**
   The `get-login` command is available in the AWS CLI starting with version 1.9.15; however, we recommend version 1.11.91 or later for recent versions of Docker (17.06 or later). You can check your AWS CLI version with the `aws --version` command. If you are using Docker version 17.06 or later, include the `--no-include-email` option after `get-login`. If you receive an Unknown options: --no-include-email error, install the latest version of the AWS CLI. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.

   ```bash
   aws ecr get-login --no-include-email --region region
   ```

4. Run the `docker login` command that was returned in the previous step. This command provides an authorization token that is valid for 12 hours.

   **Important**
   When you execute this `docker login` command, the command string can be visible to other users on your system in a process list (`ps -e`) display. Because the `docker login` command contains authentication credentials, there is a risk that other users on your system could view them this way. They could use the credentials to gain push and pull access to your repositories. If you are not on a secure system, you should consider this risk and log in interactively by omitting the `-p password` option, and then entering the password when prompted.

5. Push the image to Amazon ECR with the repositoryUri value from the earlier step.
(Optional) Clean up

When you are done experimenting with your Amazon ECR image, you can delete the repository so you are not charged for image storage.

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

Next Steps

Now that you've created a Docker image and pushed it to an Amazon ECR repository, you can begin creating your Amazon ECS resources to get a container launched. Use the following topics to continue:

- Complete the prerequisites. For more information, see Setting Up with Amazon ECS (p. 7).
- For AWS CLI walkthroughs, see Using the AWS CLI with Amazon ECS (p. 552).
- For AWS Management Console walkthroughs, see Getting Started with Amazon ECS (p. 19).
Getting Started with Amazon ECS

Get started with Amazon Elastic Container Service (Amazon ECS) by creating the Amazon ECS resources necessary to launch your first task. The Amazon ECS console provides a first-run experience that makes this easy.

In the Regions that support AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using Fargate. For more information, see AWS Fargate on Amazon ECS (p. 30). The wizard gives you the option of creating a cluster and launching a sample web application. If you already have a Docker image to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

In the Regions that don't support AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with tasks that use the EC2 launch type. The wizard gives you the option of creating a cluster and launching a sample web application. If you already have a Docker image to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

Topics
- Getting Started with Amazon ECS using Fargate (p. 19)
- Getting Started with Amazon ECS (p. 22)

Getting Started with Amazon ECS using Fargate

Get started with Amazon Elastic Container Service (Amazon ECS) by creating a task definition that uses the Fargate launch type, scheduling tasks, and configuring a cluster in the Amazon ECS console.

In the Regions that support AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using Fargate. For more information, see AWS Fargate on Amazon ECS (p. 30). The wizard gives you the option of creating a cluster and launching a sample web application. If you already have a Docker image to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

Important
For more information about the Amazon ECS first-run wizard for EC2 tasks, see Getting Started with Amazon ECS.

Complete the following tasks to get started with Amazon ECS using Fargate:

Topics
- Prerequisites (p. 19)
- Step 1: Create a Task Definition (p. 20)
- Step 2: Configure the Service (p. 20)
- Step 3: Configure the Cluster (p. 21)
- Step 4: Review (p. 21)
- Step 5: (Optional) View your Service (p. 21)

Prerequisites

Before you begin, be sure that you've completed the steps in Setting Up with Amazon ECS (p. 7) and that your AWS user has either the permissions specified in the AdministratorAccess or Amazon ECS First Run Wizard (p. 440) IAM policy example.
Step 1: 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.

2. From the navigation bar, select the US East (N. Virginia) Region.
   
   **Note**
   You can complete this first-run wizard using these steps for any Region that supports Amazon ECS using Fargate. For more information, see AWS Fargate on Amazon ECS (p. 30).
3. Configure your container definition parameters.
   
   For **Container definition**, the first-run wizard comes preloaded with the `sample-app`, `nginx`, and `tomcat-webserver` container definitions in the console. You can optionally rename the container or review and edit the resources used by the container (such as CPU units and memory limits) by choosing Edit and editing the values shown. For more information, see Container Definitions (p. 197).
   
   **Note**
   If you are using an Amazon ECR image in your container definition, be sure to use the full registry/repository:tag naming for your Amazon ECR images. For example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`.
4. For **Task definition**, the first-run wizard defines a task definition to use with the preloaded container definitions. You can optionally rename the task definition and edit the resources used by the task (such as the **Task memory** and **Task CPU** values) by choosing Edit and editing the values shown. For more information, see Task Definition Parameters (p. 195).

   Task definitions created in the first-run wizard are limited to a single container for simplicity. You can create multi-container task definitions later in the Amazon ECS console.
5. Choose Next.

Step 2: Configure the Service

In this section of the wizard, select how to configure the Amazon ECS service that is created from your task definition. A service launches and maintains a specified number of copies of the task definition in your cluster. The Amazon ECS sample application is a web-based Hello World–style application that is meant to run indefinitely. By running it as a service, it restarts if the task becomes unhealthy or unexpectedly stops.
The first-run wizard comes preloaded with a service definition, and you can see the `sample-app-service` service defined in the console. You can optionally rename the service or review and edit the details by choosing Edit and doing the following:

1. In the **Service name** field, select a name for your service.
2. In the **Number of desired tasks** field, enter the number of tasks to launch with your specified task definition.
3. In the **Security group** field, specify a range of IPv4 addresses to allow inbound traffic from, in CIDR block notation. For example, `203.0.113.0/24`.
4. (Optional) You can choose to use an Application Load Balancer with your service. When a task is launched from a service that is configured to use a load balancer, the task is registered with the load balancer. Traffic from the load balancer is distributed across the instances in the load balancer. For more information, see Introduction to Application Load Balancers.

   **Important**
   Application Load Balancers do incur cost while they exist in your AWS resources. For more information, see Application Load Balancer Pricing.

   Complete the following steps to use a load balancer with your service.

   - In the **Container to load balance** section, choose the **Load balancer listener port**. The default value here is set up for the sample application, but you can configure different listener options for the load balancer. For more information, see Service Load Balancing (p. 306).
5. Review your service settings and click **Save, Next**.

### Step 3: Configure the Cluster

In this section of the wizard, you name your cluster, and then Amazon ECS takes care of the networking and IAM configuration for you.

1. In the **Cluster name** field, choose a name for your cluster.
2. Click **Next** to proceed.

### Step 4: Review

1. Review your task definition, task configuration, and cluster configuration and click **Create** to finish. You are directed to a Launch Status page that shows the status of your launch. It describes each step of the process (this can take a few minutes to complete while your Auto Scaling group is created and populated).
2. After the launch is complete, choose **View service**.

### Step 5: (Optional) View your Service

If your service is a web-based application, such as the Amazon ECS sample application, you can view its containers with a web browser.

1. On the **Service: service-name** page, choose the **Tasks** tab.
2. Choose a task from the list of tasks in your service.
3. In the **Network** section, choose the **ENI Id** for your task. This takes you to the Amazon EC2 console where you can view the details of the network interface associated with your task, including the IPv4 Public IP address.
4. Enter the IPv4 Public IP address in your web browser and you should see a webpage that displays the Amazon ECS sample application.

Getting Started with Amazon ECS

Get started with Amazon Elastic Container Service (Amazon ECS) by creating a task definition that uses the EC2 launch type, scheduling tasks, and configuring a cluster in the Amazon ECS console. For more information, see Amazon ECS Launch Types (p. 227).

In the Regions that don’t support AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with tasks that use the EC2 launch type. The wizard gives you the option of creating a cluster and launching a sample web application. If you already have a Docker image to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

**Important**
For information about the Amazon ECS first-run wizard for Fargate tasks, see Getting Started with Amazon ECS (p. 19).

You can optionally create an Amazon Elastic Container Registry (Amazon ECR) image repository and push an image to it. For more information, see the Amazon Elastic Container Registry User Guide.

Complete the following tasks to get started with Amazon ECS:

**Topics**
- Prerequisites (p. 22)
- Step 1: Choose Your Configuration Options (p. 23)
- Step 2: Create a Task Definition (p. 24)
- Step 3: Configure the Service (p. 25)
- Step 4: Configure the Cluster (p. 25)
- Step 5: Review (p. 26)
- Step 6: (Optional) View your Service (p. 26)

**Prerequisites**

Before you begin, be sure that you’ve completed the steps in Setting Up with Amazon ECS (p. 7) and that your AWS user has either the permissions specified in the AdministratorAccess or Amazon ECS First Run Wizard (p. 440) IAM policy example.

The first-run wizard attempts to automatically create the Amazon ECS service IAM and container instance IAM role. To ensure that the first-run experience is able to create these IAM roles, one of the following must be true:
• Your user has administrator access. For more information, see Setting Up with Amazon ECS (p. 7).
• 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 these IAM roles so that they are available on the account to be used. For more information, see Amazon ECS Service Scheduler IAM Role (p. 426) and Amazon ECS Container Instance IAM Role (p. 415).

Step 1: Choose Your Configuration Options

2. Select your Amazon ECS first-run options.

I want to  ✔ Deploy a sample application onto an Amazon ECS Cluster
Amazon ECS will set up an autoscaling group and help you create a task with service management.

✔ Store container images securely with Amazon ECR
Create and manage a new private image repository and use the Docker CLI to push and pull an image. The repository is managed through AWS Identity and Access Management.

To create an Amazon ECS cluster and deploy a container application to it, check the top option. To create an Amazon ECR repository and push an image to it, which you can use in your Amazon ECS task definitions, check the bottom option. Choose Continue.

Important
If you are scoped to a Region that supports AWS Fargate then you won't see these options. For more information, see Getting Started with Amazon ECS using Fargate (p. 19).

3. If you've chosen to create an Amazon ECR repository, then complete the next two sections of the first-run wizard, Configure repository and Build, tag, and push Docker image. If you are not creating an Amazon ECR repository, skip ahead to Step 2: Create a Task Definition (p. 24).

Configure repository

A repository is where you store Docker images in Amazon ECR. Every time you push or pull an image from Amazon ECR, you specify the registry and repository location to tell Docker where to push the image to or where to pull it from.

1. Choose Get Started.
2. For Repository configuration, enter a unique name for your repository and choose Create repository.

Build, tag, and push Docker image

In this section of the wizard, you use the Docker CLI to tag an existing local image (that you have built from a Dockerfile or pulled from another registry, such as Docker Hub) and then push the tagged image to your Amazon ECR registry.
Step 2: Create a Task Definition

A task definition is like a blueprint for your application. Each time that 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.

2. From the navigation bar, select the US West (N. California) Region.
3. Configure your task definition parameters.

   The first-run wizard comes preloaded with a task definition, and you can see the simple-app container defined in the console. You can optionally rename the task definition or review and edit the resources used by the container (such as CPU units and memory limits). Choose the container name and editing the values shown (CPU units are under the Advanced options menu). Task definitions created in the first-run wizard are limited to a single container for simplicity. You can create multi-container task definitions later in the Amazon ECS console.

   For more information about what each of these task definition parameters does, see Task Definition Parameters (p. 195).
Note
If you are using an Amazon ECR image in your container definition, be sure to use the full registry/repository:tag naming for your Amazon ECR images. For example, aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest.

4. Choose Next.

Step 3: Configure the Service

In this section of the wizard, you select how you would like to configure the Amazon ECS service that is created from your task definition. A service launches and maintains a specified number of copies of the task definition in your cluster. The Amazon ECS sample application is a web-based Hello World–style application that is meant to run indefinitely. By running it as a service, it restarts if the task becomes unhealthy or unexpectedly stops.

The first-run wizard comes preloaded with a service definition, and you can see the sample-app-service service defined in the console. You can optionally rename the service or review and edit the details by choosing Edit and doing the following:

1. For Service name, select a name for your service.
2. For Desired number of tasks, enter the number of tasks to launch with your specified task definition.
3. (Optional) You can choose to use an Application Load Balancer with your service. When a task is launched from a service that is configured to use a load balancer, the task is registered with the load balancer. Traffic from the load balancer is distributed across the instances in the load balancer. For more information, see Introduction to Application Load Balancers.
   a. Choose the Application Load Balancer listener port. The default value here is set up for the sample application, but you can configure different listener options for the load balancer. For more information, see Service Load Balancing (p. 306).
   b. In the Application Load Balancer target group name field, specify a name for the target group.
   c. In the Service IAM Role section, choose either an existing Amazon ECS service (ecsServiceRole) role that you have already created, or choose Create new role to create the required IAM role for your service. For more information, see Amazon ECS Service Scheduler IAM Role (p. 426).
4. Review your service settings and choose Next step.

Step 4: Configure the Cluster

In this section of the wizard, you name your cluster. Then, Amazon ECS takes care of the networking and IAM configuration for you.

1. For Cluster name, choose a name for your cluster.
2. For EC2 instance type, choose the instance type to use for your container instances. Instance types with more CPU and memory resources can handle more tasks. For more information about the different instance types, see Amazon EC2 Instances.
3. For Number of instances, type the number of Amazon EC2 instances to launch into your cluster for tasksplacement. The more instances you have in your cluster, the more tasks you can place on them. Amazon EC2 instances incur costs while they exist in your AWS resources. For more information, see Amazon EC2 Pricing.
Note
If you created a service with more than one desired task in it that exposes container ports on to container instance ports, such as the Amazon ECS sample application, you must specify at least that many instances here.

4. Select a key pair name to use with your container instances. This is required for you to log into your instances with SSH. If you do not specify a key pair here, you cannot connect to your container instances with SSH. If you do not have a key pair, you can create one in the Amazon EC2 console. For more information, see Amazon EC2 Key Pairs.

5. (Optional) In the Security Group section, you can choose a CIDR block that restricts access to your instances. The default value (Anywhere) allows access from the entire internet.

6. In the Container instance IAM role section, choose an existing Amazon ECS container instance (ecsInstanceRole) role that you have already created, or choose Create new role to create the required IAM role for your container instances. For more information, see Amazon ECS Container Instance IAM Role (p. 415).


Step 5: Review

1. Review your task definition, task configuration, and cluster configurations and click Create to finish. You are directed to a Launch Status page that shows the status of your launch. It describes each step of the process (this can take a few minutes to complete while your Auto Scaling group is created and populated).

2. After the launch is complete, choose View service.

Step 6: (Optional) View your Service

If your service is a web-based application, such as the Amazon ECS sample application, you can view its containers with a web browser.

2. Choose a task from the list of tasks in your service.
3. In the Network section, choose the ENI Id value for your task. This takes you to the Amazon EC2 console where you can view the details of the network interface associated with your task, including the IPv4 Public IP address.
4. Enter the IPv4 Public IP address in your web browser and you should see a webpage that displays the Amazon ECS sample application.

Amazon ECS Sample App

Congratulations!

Your application is now running on a container in Amazon ECS.
Cleaning Up your Amazon ECS Resources

When you are finished experimenting with or using a particular 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.

Topics
- Scale Down Services (p. 27)
- Delete Services (p. 27)
- Deregister Container Instances (p. 27)
- Delete a Cluster (p. 28)
- Delete the AWS CloudFormation Stack (p. 28)

Scale Down Services

If your cluster contains any services, you should first scale down the desired count of tasks in these services to 0 so that Amazon ECS does not try to start new tasks on your container instances while you are cleaning up. Follow the procedure in Updating a Service (p. 350) and enter 0 in the Number of tasks field.

Alternatively, you can use the following AWS CLI command to scale down your service. Be sure to substitute the Region, cluster name, and service name for each service that you are scaling down.

```
aws ecs update-service --cluster default --service service_name --desired-count 0 --region us-west-2
```

Delete Services

Before you can delete a cluster, you must delete the services inside that cluster. After your service has scaled down to 0 tasks, you can delete it. For each service inside your cluster, follow the procedures in Deleting a Service (p. 353) to delete it.

Alternatively, you can use the following AWS CLI command to delete your services. Be sure to substitute the Region, cluster name, and service name for each service that you are deleting.

```
aws ecs delete-service --cluster default --service service_name --region us-west-2
```

Deregister Container Instances

Before you can delete a cluster, you must deregister the container instances inside that cluster. For each container instance inside your cluster, follow the procedures in Deregister a Container Instance (p. 131) to deregister it.
Alternatively, you can use the following AWS CLI command to deregister your container instances. Be sure to substitute the Region, cluster name, and container instance ID for each container instance that you are deregistering.

```
aws ecs deregister-container-instance --cluster default --container-instance container_instance_id --region us-west-2 --force
```

### Delete a Cluster

After you have removed the active resources from your Amazon ECS cluster, you can delete it.

**To delete a cluster**

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the right Region for your cluster.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, click the x in the upper-right-hand corner of the cluster to delete.

![Cluster deletion interface](image)

- Registered Container Instances: 0
- Pending tasks: 0
- Running tasks: 0

5. Choose Yes, Delete.

Alternatively, you can use the following AWS CLI command to delete your cluster. Be sure to substitute the Region and cluster name for each cluster that you are deleting.

```
aws ecs delete-cluster --cluster default --region us-west-2
```

### Delete the AWS CloudFormation Stack

If you created your Amazon ECS resources by following the console first-run wizard, then your resources are contained in an AWS CloudFormation stack. You can completely clean up all of your remaining AWS resources that are associated with this stack by deleting it. Deleting the CloudFormation stack terminates the EC2 instances, removes the Auto Scaling group, deletes any Elastic Load Balancing load balancers, and removes the Amazon VPC subnets and internet gateway associated with the cluster.

**To delete the AWS CloudFormation stack**

2. From the navigation bar, select the Region in which your cluster was created.
3. Select the stack that is associated with your Amazon ECS resources. The Stack Name value starts with EC2ContainerService-default.
4. Choose Delete Stack and then choose Yes, Delete.
AWS Fargate on 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.

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

AWS Fargate with Amazon ECS is currently only available in the following Regions:

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
</tr>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
</tr>
<tr>
<td>EU (Ireland)</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>EU (London)</td>
<td>eu-west-2</td>
</tr>
<tr>
<td>EU (Frankfurt)</td>
<td>eu-central-1</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
</tr>
</tbody>
</table>

The following walkthroughs help you get started using AWS Fargate with Amazon ECS:

- *Getting Started with Amazon ECS* (p. 19)
- the section called “Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI” (p. 552)
- the section called “Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI” (p. 458)

Task Definitions

Tasks that use the Fargate launch type do not support all of the task definition parameters that are available. Some parameters are not 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
- pidMode
- placementConstraints
- privileged
- systemControls

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 `add` parameter is not supported. The `devices`, `sharedMemorySize`, and `tmpfs` parameters are not supported. For more information, see Linux Parameters (p. 214).
- `volumes` - Fargate tasks only support bind mount host volumes, so the `dockerVolumeConfiguration` parameter is not supported. For more information, see Volumes (p. 220).

To ensure that your task definition validates for use with the Fargate launch type, 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.

## Network Mode

Fargate task definitions require that the network mode is set to awsvpc. The awsvpc network mode provides each task with its own elastic network interface. For more information, see Task Networking with the awsvpc Network Mode (p. 242).

A network configuration is also required when creating a service or manually running tasks. For more information, see Task Networking (p. 34).

## Task CPU and Memory

Fargate task definitions require that you specify CPU and memory at the task level. Although you can also specify CPU and memory at the container level for Fargate tasks, this is optional. 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.

<table>
<thead>
<tr>
<th>CPU value</th>
<th>Memory value</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 (.25 vCPU)</td>
<td>0.5 GB, 1 GB, 2 GB</td>
</tr>
</tbody>
</table>
### CPU value

<table>
<thead>
<tr>
<th>CPU value</th>
<th>Memory value</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 (.5 vCPU)</td>
<td>1 GB, 2 GB, 3 GB, 4 GB</td>
</tr>
<tr>
<td>1024 (1 vCPU)</td>
<td>2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB</td>
</tr>
<tr>
<td>2048 (2 vCPU)</td>
<td>Between 4 GB and 16 GB in 1-GB increments</td>
</tr>
<tr>
<td>4096 (4 vCPU)</td>
<td>Between 8 GB and 30 GB in 1-GB increments</td>
</tr>
</tbody>
</table>

---

### Logging

Fargate task definitions only support the `awslogs` and `splunk` 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:

```json
"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 [Using the awslogs Log Driver](p. 245).

For more information about using the `splunk` log driver in a task definition, see [Example: splunk LogDriver](p. 262).

### Amazon ECS Task Execution IAM Role

There is an optional task execution IAM role that you can specify with Fargate to allow your Fargate tasks to make API calls to Amazon ECR. The API calls pull container images as well as calling CloudWatch to store container application logs. For more information, see [Amazon ECS Task Execution IAM Role](p. 418).

### Example Task Definition

The following is an example task definition that sets up a web server using the Fargate launch type:

```json
{
  "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! 

Your application is now running on a container in Amazon ECS. </h2> <h2>Congratulations! 

Your application is now running on a container in Amazon ECS. </h2>
" >> /usr/local/apache2/htdocs/index.html && httpd-foreground"
      ],
      "entryPoint": [
        "sh",
        "-c"
      ],
      "essential": true,
      "memory": 1024,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ],
      "cpu": 1024
    }
  ]
}
```
Task Storage

When provisioned, each Fargate task receives the following storage. Task storage is ephemeral. After a Fargate task stops, the storage is deleted.

- 10 GB of Docker layer storage
- An additional 4 GB for volume mounts. This can be mounted and shared among containers using the volumes, mountPoints, and volumesFrom parameters in the task definition.

**Note**
The `host` and `sourcePath` parameters are not supported.

For more information about Amazon ECS default service limits, see Amazon ECS Service Limits (p. 587).

The following shows a snippet of a task definition where two containers are sharing a single volume:
"sourceVolume": "database_scratch"
}],
"name": "database2",
}
],
"volumes": [
{
"name": "database_scratch"
}
]

Tasks and Services

After you have your Fargate task definition prepared, there are some decisions to make when creating your service.

Task Networking

Tasks using the Fargate launch type 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, the private subnet requires a NAT gateway be attached to route requests to the internet. For more information, see Task Networking with the awsvpc Network Mode (p. 242).

The following is an example of the networkConfiguration section for a Fargate service:

```
"networkConfiguration": {
"awsvpcConfiguration": {
"assignPublicIp": "ENABLED",
"securityGroups": [ "sg-12345678" ],
"subnets": [ "subnet-12345678" ]
}
}
```

Services with tasks that use the awsvpc network mode (for example, those with the Fargate launch type) only support Application Load Balancers and Network Load Balancers. Classic Load Balancers are not supported. Also, 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 elastic network interface, not an Amazon EC2 instance. For more information, see Service Load Balancing (p. 306).

Private Registry Authentication

Fargate tasks can authenticate with private image registries, including Docker Hub, using basic authentication. When you enable private registry authentication, you can use private Docker images in your task definitions.
To use private registry authentication, you create a secret with AWS Secrets Manager containing the credentials for your private registry. Then, within your container definition, you specify `repositoryCredentials` with the full ARN of the secret that you created. The following snippet of a task definition shows the required parameters:

```json
"containerDefinitions": [
    {
        "image": "private-repo/private-image",
        "repositoryCredentials": {
        }
    }
]
```

For more information, see Private Registry Authentication for Tasks (p. 251).

## Clusters

Clusters can contain tasks using both the Fargate and EC2 launch types. When viewing your clusters in the AWS Management Console, Fargate and EC2 task counts are displayed separately.

For more information about Amazon ECS clusters, including a walkthrough for creating a cluster, see Amazon ECS Clusters (p. 55).

## Fargate Task Retirement

A Fargate task is scheduled to be retired when AWS detects the irreparable failure of the underlying hardware hosting the task or if a security issue needs to be patched. Most security patches are handled transparently without requiring any action on your part or having to restart your tasks. But for certain issues, we may require that the task be restarted.

When a task reaches its scheduled retirement date, it is stopped or terminated by AWS. If the task is part of a service, then the task is automatically stopped and the service scheduler starts a new one to replace it. If you are using standalone tasks, then you receive notification of the task retirement. For more information, see Task Retirement (p. 287).
AWS Fargate 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.

New 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. Security updates and patches are deployed automatically for your Fargate tasks. If a security issue is found that affects a platform version, AWS patches the platform version. In some cases, you may be notified that your Fargate tasks have been scheduled for retirement. For more information, see Task Retirement (p. 287).

Topics
- Platform Version Considerations (p. 36)
- Available AWS Fargate Platform Versions (p. 36)

Platform Version Considerations

The following should be considered when specifying a platform version:

- When specifying a platform version, you can use either the version number (for example, 1.2.0) or LATEST.
- To use a specific platform version, specify the version number when creating or updating your service. If you specify LATEST, your tasks use the most current platform version available, which may not be the most recent platform version.
- If you have a service with running tasks and want to update their platform version, you can update your service, specify a new platform version, and choose Force new deployment. Your tasks are redeployed with the latest platform version. For more information, see Updating a Service (p. 350).
- 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.

Available AWS Fargate Platform Versions

The following is a list of the platform versions currently available:

Fargate Platform Version-1.3.0
- 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 Fargate Task Recycling (p. 288).
- Beginning on March 27, 2019, any new Fargate task that is 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 (p. 225), Container Dependency (p. 216), and ContainerTimeouts (p. 217).
- 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 Specifying Sensitive Data (p. 254).
- 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 Specifying Sensitive Data (p. 254).
• 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 (p. 207).

Fargate Platform Version-1.2.0
• Added support for private registry authentication using AWS Secrets Manager. For more information, see Private Registry Authentication for Tasks (p. 251).

Fargate Platform Version-1.1.0
• Added support for the Amazon ECS task metadata endpoint. For more information, see Amazon ECS Task Metadata Endpoint (p. 172).
• Added support for Docker health checks in container definitions. For more information, see Health Check (p. 201).
• Added support for Amazon ECS service discovery. For more information, see Service Discovery (p. 328).

Fargate Platform Version-1.0.0
• Based on Amazon Linux 2017.09.
• Initial release.
Working with GPUs on Amazon ECS

Amazon ECS supports workloads that take advantage of GPUs by enabling you to create clusters with GPU-enabled container instances. Amazon EC2 GPU-based container instances using the p2 and p3 instance types provide access to NVIDIA GPUs. For more information, see Linux Accelerated Computing Instances in the Amazon EC2 User Guide for Linux Instances.

Amazon ECS provides a GPU-optimized AMI that comes ready with pre-configured NVIDIA kernel drivers and a Docker GPU runtime. For more information, see Amazon ECS-optimized AMIs (p. 62).

You can designate a number of GPUs in your task definition for task placement consideration at a container level. Amazon ECS will schedule to available GPU-enabled container instances 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 and Amazon EC2 P3 Instances.

<table>
<thead>
<tr>
<th>Instance type</th>
<th>GPUs</th>
<th>GPU Memory (GiB)</th>
<th>vCPUs</th>
<th>Memory (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2.xlarge</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>61</td>
</tr>
<tr>
<td>p2.8xlarge</td>
<td>8</td>
<td>96</td>
<td>32</td>
<td>488</td>
</tr>
<tr>
<td>p2.16xlarge</td>
<td>16</td>
<td>192</td>
<td>64</td>
<td>732</td>
</tr>
<tr>
<td>p3.2xlarge</td>
<td>1</td>
<td>16</td>
<td>8</td>
<td>61</td>
</tr>
<tr>
<td>p3.8xlarge</td>
<td>4</td>
<td>64</td>
<td>32</td>
<td>244</td>
</tr>
<tr>
<td>p3.16xlarge</td>
<td>8</td>
<td>128</td>
<td>64</td>
<td>488</td>
</tr>
<tr>
<td>p3dn.24xlarge</td>
<td>8</td>
<td>256</td>
<td>96</td>
<td>768</td>
</tr>
</tbody>
</table>

Topics
- Considerations for Working with GPUs (p. 38)
- Specifying GPUs in Your Task Definition (p. 39)

Considerations for Working with GPUs

Before you begin working with GPUs on Amazon ECS, be aware of the following considerations:

- Your clusters can contain a mix of GPU and non-GPU container instances.
- When running a task or creating a service, you can use instance type attributes when configuring task placement constraints to ensure which of your container instances the task is launched on. This will enable you to effectively use your resources. For more information, see Amazon ECS Task Placement (p. 274).

The following example launches a task on a p2.xlarge container instance in your default cluster.

```
aws ecs run-task --cluster default --task-definition ecs-gpu-task-def \
```
Specifying GPUs in Your Task Definition

To take advantage of the GPUs on a container instance and the Docker GPU runtime, ensure you designate the number of GPUs your container requires in the task definition. As GPU-enabled containers are placed, the Amazon ECS container agent will pin the desired number of physical GPUs to the appropriate container. The number of GPUs reserved for all containers in a task should not exceed the number of available GPUs on the container instance the task is launched on. For more information, see Creating a Task Definition (p. 188).

**Important**
If your GPU requirements are not specified in the task definition, the task will use 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 2 GPUs, runs the `nvidia-smi` utility and then exits.

```
{
    "containerDefinitions": [
        {
            "memory": 80,
            "essential": true,
            "name": "gpu",
            "image": "nvidia/cuda:9.0-base",
            "resourceRequirements": [
                {
                    "type": "GPU",
                    "value": "2"
                }
            ],
            "command": [
                "sh",
                "-c",
                "nvidia-smi"
            ],
            "cpu": 100
        }
    }
}
```
```json
{
    "family": "example-ecs-gpu"
}
```
Tutorials for Amazon ECS

The following tutorials show you how to perform common tasks when using Amazon ECS.

Topics
- Tutorial: Creating a Service Using a Blue/Green Deployment (p. 41)
- Tutorial: Continuous Deployment with CodePipeline (p. 49)

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 a Service (p. 340).

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.

Topics
- Prerequisites (p. 41)
- Step 1: Create an Application Load Balancer (p. 41)
- Step 2: Create an Amazon ECS Cluster (p. 42)
- Step 3: Register a Task Definition (p. 42)
- Step 4: Create an Amazon ECS Service (p. 43)
- Step 5: Create the AWS CodeDeploy Resources (p. 44)
- Step 5: Create and Monitor an CodeDeploy Deployment (p. 46)
- Step 6: Clean Up (p. 48)

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 Setting Up with Amazon ECS (p. 7) have been completed.
- Your AWS user has the required permissions specified in the Amazon ECS First Run Wizard (p. 440) IAM policy example.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

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.
Step 2: Create an Amazon ECS Cluster

Use the create-cluster command to create a cluster named tutorial-bluegreen to use.

```
aws ecs create-cluster --cluster-name tutorial-bluegreen --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
```

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.

```
aws ecs register-task-definition --task-definition-name tutorial-bluegreen --network-configuration awsvpc --region us-east-1
```

The output includes the ARN of the task definition, with the following format:

```
```
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 (p. 418).

```json
{
  "family": "tutorial-task-def",
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "sample-app",
      "image": "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",
  "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"
}
```

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.

```json
{
  "cluster": "tutorial-bluegreen",
  "serviceName": "service-bluegreen",
  "taskDefinition": "tutorial-task-def",
  "loadBalancers": [
    {
      "targetGroupArn": "arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/bluegreentarget1/209a844cd01825a4",
      "containerName": "sample-app",
      "containerPort": 80
    }
  ]
}
```

API Version 2014-11-13
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.

   ```bash
   aws deploy create-application --application-name tutorial-bluegreen
   --compute-platform ECS --region us-east-1
   ```

   The output includes the application ID, with the following format:

   ```bash
   
   ```

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.

   ```bash
   aws elbv2 create-target-group --name bluegreentarget2 --protocol HTTP --port 80
   --target-type ip --vpc-id "vpc-0b6d82c67d8012a1" --region us-east-1
   ```

   The output includes the ARN for the target group, with the following format:

   ```bash
   arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/bluegreentarget2/708d384187a3cfdc
   ```

3. Use the `create-deployment-group` command to create an CodeDeploy deployment group.

   ```bash
   ```

   ```bash
   ```

   ```bash
   ```
First, create a file named `tutorial-deployment-group.json` with the following contents. This example uses the resource that you created.

```json
{
  "applicationName": "tutorial-bluegreen",
  "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"
      }
    ]
  }
}
```

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:

```json
{
  "API Version 2014-11-13"
  45
}
Step 5: 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.

   ```yaml
   version: 0.0
   Resources:
   - TargetService:
       Type: AWS::ECS::Service
       Properties:
       LoadBalancerInfo:
       ContainerName: "sample-app"
       ContainerPort: 80
       PlatformVersion: "LATEST"
   
   b. Use the `s3 mb` command to create an Amazon S3 bucket for the AppSpec file.

   ```bash
   aws s3 mb s3://tutorial-bluegreen
   ```

   c. Use the `s3 cp` command to upload the AppSpec file to the Amazon S3 bucket.

   ```bash
   aws s3 cp ./AppSpec.yaml s3://tutorial-bluegreen/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.

   ```json
   {
   "applicationName": "tutorial-bluegreen",
   "deploymentGroupName": "tutorial-bluegreen-dg",
   "revision": {
   "revisionType": "S3",
   "s3Location": {
   "bucket": "tutorial-bluegreen",
   "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: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: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": 1543431488.793,
        "endTime": 1543431489.222,
        "status": "Succeeded"
      }
    ]
  }
}
```
Step 6: 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.

   ```bash
   aws deploy delete-deployment-group --application-name tutorial-bluegreen --deployment-group-name tutorial-bluegreen-dg --region us-east-1
   ```

2. Use the `delete-application` command to delete the CodeDeploy application.

   ```bash
   aws deploy delete-application --application-name tutorial-bluegreen --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.

   ```bash
   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.

   ```bash
   aws ecs delete-cluster --cluster tutorial-bluegreen --region us-east-1
   ```
5. Use the `s3 rm` command to delete the AppSpec file from the Amazon S3 bucket.

   ```bash
   aws s3 rm s3://tutorial-bluegreen/appspec.yaml
   ```

6. Use the `s3 rb` command to delete the Amazon S3 bucket.

   ```bash
   aws s3 rb s3://tutorial-bluegreen
   ```

7. Use the `delete-load-balancer` command to delete the Application Load Balancer.

   ```bash
   aws elbv2 delete-load-balancer --load-balancer-arn
e5ba6279c16e642 --region us-east-1
   ```

8. Use the `delete-target-group` command to delete the two Application Load Balancer target groups.

   ```bash
   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/
   bluegreentarget2/70d39e18a9c0f0d --region us-east-1
   ```

---

**Tutorial: Continuous Deployment with CodePipeline**

This tutorial helps you to create a complete, end-to-end continuous deployment (CD) pipeline with Amazon ECS with CodePipeline.

**Prerequisites**

There are a few resources that you must have in place before you can use this tutorial to create your CD pipeline. Here are the things you need to get started:

**Note**

All of these resources should be created within the same AWS Region.

- A source control repository (this tutorial uses CodeCommit) with your Dockerfile and application source. For more information, see Create an CodeCommit Repository in the AWS CodeCommit User Guide.

- A Docker image repository (this tutorial uses Amazon ECR) that contains an image you have built from your Dockerfile and application source. For more information, see Creating a Repository and Pushing an Image in the Amazon Elastic Container Registry User Guide.

- An Amazon ECS task definition that references the Docker image hosted in your image repository. For more information, see Creating a Task Definition in the Amazon Elastic Container Service Developer Guide.

- An Amazon ECS cluster that is running a service that uses your previously mentioned task definition. For more information, see Creating a Cluster and Creating a Service in the Amazon Elastic Container Service Developer Guide.
After you have satisfied these prerequisites, you can proceed with the tutorial and create your CD pipeline.

**Step 1: Add a Build Specification File to Your Source Repository**

This tutorial uses CodeBuild to build your Docker image and push the image to Amazon ECR. Add a buildspec.yml file to your source code repository to tell CodeBuild how to do that. The example build specification below does the following:

- **Pre-build stage:**
  - Log in to Amazon ECR.
  - Set the repository URI to your ECR image and add an image tag with the first seven characters of the Git commit ID of the source.

- **Build stage:**
  - Build the Docker image and tag the image both as `latest` and with the Git commit ID.

- **Post-build stage:**
  - Push the image to your ECR repository with both tags.
  - Write a file called `imagedefinitions.json` in the build root that has your Amazon ECS service's container name and the image and tag. The deployment stage of your CD pipeline uses this information to create a new revision of your service's task definition, and then it updates the service to use the new task definition. The `imagedefinitions.json` file is required for the CodeDeploy ECS job worker.

```yaml
version: 0.2
phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '
' "{{"name":"hello-world","imageUri":"%s"}}' $REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
artifacts:
  files: imagedefinitions.json
```

The build specification was written for the following task definition, used by the Amazon ECS service for this tutorial. The `REPOSITORY_URI` value corresponds to the image repository (without any image tag),
and the `hello-world` value near the end of the file corresponds to the container name in the service's task definition.

```json
{
  "taskDefinition": {
    "family": "hello-world",
    "containerDefinitions": [
      {
        "name": "hello-world",
        "image": "012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world:6a57b99",
        "cpu": 100,
        "portMappings": [
          {
            "protocol": "tcp",
            "containerPort": 80,
            "hostPort": 80
          }
        ],
        "memory": 128,
        "essential": true
      }
    ]
  }
}
```

To add a `buildspec.yml` file to your source repository

1. Open a text editor and then copy and paste the build specification above into a new file.
2. Replace the `REPOSITORY_URI` value (`012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world`) with your Amazon ECR repository URI (without any image tag) for your Docker image. Replace `hello-world` with the container name in your service's task definition that references your Docker image.
3. Commit and push your `buildspec.yml` file to your source repository.
   a. Add the file.
      
      ```
      git add .
      ```
   b. Commit the change.
      
      ```
      git commit -m "Adding build specification."
      ```
   c. Push the commit.
      
      ```
      git push
      ```

**Step 2: Creating Your Continuous Deployment Pipeline**

Use the CodePipeline wizard to create your pipeline stages and connect your source repository to your ECS service.

To create your pipeline

2. On the **Welcome** page, choose **Create pipeline**.
Step 3: Add Amazon ECR Permissions to the CodeBuild Role

The CodePipeline wizard created an IAM role for the CodeBuild build project, called `code-build-build-project-name-service-role`. For this tutorial, the name is `code-build-hello-world-service-role`. Because the `buildspec.yml` file makes calls to Amazon ECR API operations, the role must have a policy that allows permissions to make these Amazon ECR calls. The following procedure helps you attach the proper permissions to the role.
To add Amazon ECR permissions to the CodeBuild role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose Roles.
3. In the search box, type code-build- and choose the role that was created by the CodePipeline wizard. For this tutorial, the role name is code-build-hello-world-service-role.
4. On the Summary page, choose Attach policy.
5. Select the box to the left of the AmazonEC2ContainerRegistryPowerUser policy, and choose Attach policy.

Step 4: Test Your Pipeline

Your pipeline should have everything for running an end-to-end native AWS continuous deployment. Now, test its functionality by pushing a code change to your source repository.

To test your pipeline

1. Make a code change to your configured source repository, commit, and push the change.
3. Choose your pipeline from the list.
4. Watch the pipeline progress through its stages. Your pipeline should complete and your Amazon ECS service runs the Docker image that was created from your code change.
hello-world View pipeline history

View progress and manage your pipeline.

Edit Release change

Source

Source AWS CodeCommit

Succeeded 6 min ago
12 total

Source: Pickle Rick!!!

Build

CodeBuild

AWS CodeBuild

Succeeded 3 min ago
Details

Source: Pickle Rick!!!

Staging

hello-world

Amazon ECS

Succeeded 1 min ago
Details

Source: Pickle Rick!!!
Amazon ECS Clusters

An Amazon ECS cluster is a logical grouping of tasks or services. If you are running tasks or services that use the EC2 launch type, a cluster is also a grouping of container instances. When you first use Amazon ECS, a default cluster is created for you, but you can create multiple clusters in an account to keep your resources separate.

The following are general concepts about Amazon ECS clusters.

- Clusters are Region-specific.
- Clusters can contain tasks using both the Fargate and EC2 launch types. For more information about launch types, see Amazon ECS Launch Types (p. 227).
- For tasks using the EC2 launch type, clusters can contain multiple different container instance types, but each container instance may only be part of one cluster at a time.
- You can create custom IAM policies for your clusters to allow or restrict user access to specific clusters. For more information, see the Clusters (p. 443) section in Amazon ECS IAM Policy Examples (p. 439).

Topics
- Creating a Cluster (p. 55)
- Scaling a Cluster (p. 57)
- Deleting a Cluster (p. 59)

Creating a Cluster

You can create an Amazon ECS cluster using the AWS Management Console, as described in this topic. Before you begin, be sure that you’ve completed the steps in Setting Up with Amazon ECS (p. 7). If you are launching tasks with the EC2 launch type, you can register container instances into the cluster after creating it.

Note
This cluster creation wizard provides a simple way to create the resources that are needed by an Amazon ECS cluster. It also lets you customize several common cluster configuration options. However, this wizard does not allow you to customize every resource option. For example, you can't use the wizard to customize the container instance AMI ID. If your requirements extend beyond what is supported in this wizard, consider using our reference architecture at https://github.com/awslabs/ecs-refarch-cloudformation.

Do not attempt to modify the underlying resources directly after they are created by the wizard.

To create a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
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. For Select cluster compatibility, choose one of the following options and then choose Next Step:
   - Networking only—With this option, you can launch a cluster of tasks using the Fargate launch type. The Fargate launch type allows you to run your containerized applications without the need to provision and manage the backend infrastructure. When you register your task definition, Fargate launches the container for you.
Creating a Cluster

- **EC2 Linux + Networking** – With this option you can launch a cluster of tasks using the EC2 launch type and Linux containers. The EC2 launch type allows you to run your containerized applications on a cluster of Amazon EC2 instances that you manage.

- **EC2 Windows + Networking** – With this option you can launch a cluster of tasks using the EC2 launch type using Windows containers. The EC2 launch type allows you to run your containerized applications on a cluster of Amazon EC2 instances that you manage. For more information, see Windows Containers (p. 610).

If you chose the **Networking only** cluster template, continue with the following steps. Otherwise, you can skip to the next section.

**Using the Networking only cluster template**

1. On the **Configure cluster** page, enter a **Cluster name**. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
2. In the **Networking** section, configure the VPC for your cluster. You can keep the default settings, or you can modify these settings with the following steps.
   a. (Optional) If you choose to create a new VPC, for **CIDR Block**, select a CIDR block for your VPC. For more information, see Your VPC and Subnets in the Amazon VPC User Guide.
   b. For **Subnets**, select the subnets to use for your VPC. You can keep the default settings, or you can modify them to meet your needs.
3. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see Tagging Your Amazon ECS Resources.
4. Choose **Create**.

If you chose the **EC2 Linux + Networking** or **EC2 Windows + Networking** templates, complete the following steps.

**Using the EC2 Linux + Networking or EC2 Windows + Networking cluster template**

1. For **Cluster name**, enter a name for your cluster. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
2. (Optional) To create a cluster with no resources, choose **Create an empty cluster**, Create.
3. For **Provisioning model**, choose one of the following instance types:
   - **On-Demand Instance** – With On-Demand Instances, you pay for compute capacity by the hour with no long-term commitments or upfront payments.
   - **Spot** – Spot Instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. For more information, see Spot Instances.
     
     **Note**
     Spot Instances are subject to possible interruptions. We recommend that you avoid Spot Instances for applications that can't be interrupted. For more information, see Spot Instance Interruptions.

4. For Spot Instances, do the following; otherwise, skip to the next step.
   a. For **Spot Instance allocation strategy**, choose the strategy that meets your needs. For more information, see Spot Fleet Allocation Strategy.
   b. For **Maximum bid price (per instance/hour)**, specify a bid price. If your bid price is lower than the Spot price for the instance types that you selected, your Spot Instances are not launched.
5. For **EC2 instance type**, choose the Amazon EC2 instance type for your container instances. The instance type that you select determines the EC2 AMI IDs and resources available for your tasks. For
Scaling a Cluster

If you have a cluster that contains Amazon EC2 container instances, the following helps you scale the number of Amazon EC2 instances in your cluster.

GPU workloads, choose an instance type from the P2 or P3 instance family. For more information, see Working with GPUs on Amazon ECS (p. 38).

6. For **Number of instances**, choose the number of EC2 instances to launch into your cluster. These instances are launched using the latest Amazon ECS-optimized Amazon Linux AMI required by the instance type you chose. For more information, see Amazon ECS-optimized AMIs (p. 62).

7. For **EC2 AMI ID**, choose the Amazon ECS-optimized AMI for your container instances. The available AMIs will be determined by the Region and EC2 instance type you chose. For more information, see Amazon ECS-optimized AMIs (p. 62).

8. For **EBS storage (GiB)**, choose the size of the Amazon EBS volume to use for data storage on your container instances. You can increase the size of the data volume to allow for greater image and container storage.

9. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for SSH access. If you do not specify a key pair, you cannot connect to your container instances with SSH. For more information, see Amazon EC2 Key Pairs in the Amazon EC2 User Guide for Linux Instances.

10. In the **Networking** section, configure the VPC to launch your container instances into. By default, the cluster creation wizard creates a new VPC with two subnets in different Availability Zones, and a security group open to the internet on port 80. This is a basic setup that works well for an HTTP service. However, you can modify these settings by following the substeps below.

    a. For **VPC**, create a new VPC, or select an existing VPC.
    b. (Optional) If you chose to create a new VPC, for **CIDR Block**, select a CIDR block for your VPC. For more information, see Your VPC and Subnets in the Amazon VPC User Guide.
    c. For **Subnets**, select the subnets to use for your VPC. If you chose to create a new VPC, you can keep the default settings or you can modify them to meet your needs. If you chose to use an existing VPC, select one or more subnets in that VPC to use for your cluster.
    d. For **Security group**, select the security group to attach to the container instances in your cluster. If you choose to create a new security group, you can specify a CIDR block to allow inbound traffic from. The default port 0.0.0.0/0 is open to the internet. You can also select a single port or a range of contiguous ports to open on the container instance. For more complicated security group rules, you can choose an existing security group that you have already created.

       **Note**
       You can also choose to create a new security group and then modify the rules after the cluster is created. For more information, see Amazon EC2 Security Groups for Linux Instances in the Amazon EC2 User Guide for Linux Instances.
    
    e. In the **Container instance IAM role** section, select the IAM role to use with your container instances. If your account has the ecsinstanceRole that is created for you in the console first-run wizard, it is selected by default. If you do not have this role in your account, you can choose to create the role, or you can choose another IAM role to use with your container instances.

       **Important**
       If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent does not connect to your cluster. For more information, see Amazon ECS Container instance IAM Role (p. 415).
    
    f. If you chose the Spot Instance type earlier, the **Spot Fleet Role IAM role** section indicates that an IAM role ecsSpotFleetRole is created.
    
    g. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see Tagging Your Amazon ECS Resources.
    
    h. Choose **Create**.
Note
Clusters with Fargate tasks can be scaled using Service Auto Scaling. For more information, see Service Auto Scaling (p. 321).

If your cluster was created with the console first-run experience after November 24, 2015, then the Auto Scaling group associated with the AWS CloudFormation stack created for your cluster can be scaled up or down to add or remove container instances. You can perform this scaling operation from within the Amazon ECS console.

If your cluster was not created with the console first-run experience after November 24, 2015, then you cannot scale your cluster from the Amazon ECS console. However, you can still modify existing Auto Scaling groups associated with your cluster in the Auto Scaling console. If you do not have an Auto Scaling group associated with your cluster, you can create one from an existing container instance. For more information, see Creating an Auto Scaling Group Using an EC2 Instance in the Amazon EC2 Auto Scaling User Guide. You can also manually launch or terminate container instances from the Amazon EC2 console; for more information see Launching an Amazon ECS Container Instance (p. 101).

To scale a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, choose the Region in which your cluster exists.
3. In the navigation pane, choose Clusters and select the cluster to scale.

   If a Scale ECS Instances button appears, then you can scale your cluster in the next step. If not, you must manually adjust your Auto Scaling group to scale up or down your instances, or you can manually launch or terminate your container instances in the Amazon EC2 console.

5. Choose Scale ECS Instances.
6. For Desired number of instances, enter the number of instances to which to scale your cluster and choose Scale.

   Note
   If you reduce the number of container instances in your cluster, randomly selected container instances are terminated until the desired count is achieved, and any tasks that are running on terminated instances are stopped.
Deleting a Cluster

If you are finished using a cluster, you can delete it. When you delete a cluster in the Amazon ECS console, the associated resources that are deleted with it vary depending on how the cluster was created. Step 5 (p. 59) of the following procedure changes based on that condition.

If your cluster was created with the console first-run experience described in Getting Started with Amazon ECS (p. 19) after November 24, 2015, or the cluster creation wizard described in Creating a Cluster (p. 55), then the AWS CloudFormation stack that was created for your cluster is also deleted when you delete your cluster.

If your cluster was created manually (without the cluster creation wizard) or with the console first-run experience before November 24, 2015, then you must deregister (or terminate) any container instances associated with the cluster before you can delete it. For more information, see Deregister a Container Instance (p. 131). In this case, after the cluster is deleted, you should delete any remaining AWS CloudFormation stack resources or Auto Scaling groups associated with the cluster to avoid incurring any future charges for those resources. For more information, see Delete the AWS CloudFormation Stack (p. 28).

To delete a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
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.
   
   Note
   If your cluster has registered container instances, you must deregister or terminate them. For more information, see Deregister a Container Instance (p. 131).

5. Choose Delete Cluster. You see one of two confirmation prompts:

   • **Deleting the cluster also deletes the CloudFormation stack**
     
     EC2ContainerService-\textit{cluster\_name} – Deleting this cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

   • **Deleting the cluster does not affect CloudFormation resources** – Deleting this cluster does not clean up any resources that are associated with the cluster, including Auto Scaling groups, VPCs, or load balancers. Also, any container instances that are registered with this cluster must be deregistered or terminated before you can delete the cluster. For more information, see Deregister a Container Instance (p. 131). You can visit the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation/ to update or delete any of these resources. For more information, see Delete the AWS CloudFormation Stack (p. 28).
Amazon ECS Container Instances

An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into a cluster. When you run tasks with Amazon ECS, your tasks using the EC2 launch type are placed on your active container instances.

**Note**
Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply.

**Topics**
- Container Instance Concepts (p. 60)
- Container Instance Lifecycle (p. 61)
- Check the Instance Role for Your Account (p. 61)
- Amazon ECS-optimized AMIs (p. 62)
- Retrieving Amazon ECS-Optimized AMI Metadata (p. 91)
- Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications (p. 97)
- Launching an Amazon ECS Container Instance (p. 101)
- Bootstrapping Container Instances with Amazon EC2 User Data (p. 105)
- Connect to Your Container Instance (p. 113)
- Using CloudWatch Logs with Container Instances (p. 114)
- Container Instance Draining (p. 122)
- Container Instance Memory Management (p. 124)
- Managing Container Instances Remotely (p. 125)
- Starting a Task at Container Instance Launch Time (p. 128)
- Deregister a Container Instance (p. 131)

**Container Instance Concepts**

- Your container instance must be running the Amazon ECS container agent to register into one of your clusters. If you are using an Amazon ECS-optimized AMI, the agent is already installed. To use a different operating system, install the agent. For more information, see Amazon ECS Container Agent (p. 133).

- Because the Amazon ECS container agent makes calls to Amazon ECS on your behalf, you must launch container instances with an IAM role that authenticates to your account and provides the required resource permissions. For more information, see Amazon ECS Container Instance IAM Role (p. 415).

- If any of the containers associated with your tasks require external connectivity, you can map their network ports to ports on the host Amazon ECS container instance so they are reachable from the internet. Your container instance security group must allow inbound access to the ports you want to expose. For more information, see Create a Security Group in the Amazon VPC Getting Started Guide.

- We strongly recommend launching your container instances inside a VPC, because Amazon VPC delivers more control over your network and offers more extensive configuration capabilities. For more information, see Amazon EC2 and Amazon Virtual Private Cloud in the Amazon EC2 User Guide for Linux Instances.

- Container instances need external network access to communicate with the Amazon ECS service endpoint, so if 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 Elastic Container Service Developer Guide

Container Instance Lifecycle

When the Amazon ECS container agent registers an instance into your cluster, the container 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) an Amazon ECS 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** (p. 146).

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 **Container Instance Draining** (p. 122).

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.

Check the Instance Role for Your Account

The Amazon ECS container agent makes calls to the Amazon ECS APIs on your behalf. Container instances that run the agent require an IAM policy and role for the service to know that the agent belongs to you.
In most cases, the Amazon ECS instance role is automatically created for you in the console first-run experience. You can use the following procedure to check and see if your account already has an Amazon ECS service role.

To check for the `ecsInstanceRole` in the 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, choose Roles.
3. Search the list of roles for `ecsInstanceRole`. If the role exists, you do not need to create it. If the role does not exist, follow the procedures in Amazon ECS Container Instance IAM Role (p. 415) to create the role.

Amazon ECS-optimized AMIs

The basic Amazon Elastic Container Service (Amazon ECS) container instance specification consists of the following:

Required

- A modern Linux distribution running at least version 3.10 of the Linux kernel.
- The Amazon ECS container agent (preferably the latest version). For more information, see Amazon ECS Container Agent (p. 133).
- A Docker daemon running at least version 1.9.0, and any Docker runtime dependencies. For more information, see Check runtime dependencies in the Docker documentation.

Note

For the best experience, we recommend the Docker version that ships with and is tested with the corresponding Amazon ECS agent version that you are using. For more information, see Amazon ECS Container Agent Versions (p. 142).

Recommended

- An initialization and nanny process to run and monitor the Amazon ECS agent. The Amazon ECS-optimized AMIs use the `ecs-init` RPM to manage the agent. For more information, see the `ecs-init` project on GitHub.

The Amazon ECS-optimized AMIs are preconfigured with these requirements and recommendations. We recommend that you use the Amazon ECS-optimized Amazon Linux 2 AMI for your container instances unless your application requires a specific operating system or a Docker version that is not yet available in that AMI.

Amazon ECS vends AMIs that are optimized for the service in the following variants.

- **Amazon ECS-optimized Amazon Linux 2 AMI** – Recommended for launching your Amazon ECS container instances in most cases.
- **Amazon ECS-optimized Amazon Linux 2 (arm64) AMI** – Recommended for launching your Amazon ECS container instances when using the Amazon EC2 A1 instance type, which is powered by Arm-based AWS Graviton Processors. For more information, see General Purpose Instances in the Amazon EC2 User Guide for Linux Instances.
- **Amazon ECS GPU-optimized AMI** – Recommended for launching your Amazon ECS container instances when working with GPU workloads. For more information, see Working with GPUs on Amazon ECS (p. 38).
• **Amazon ECS-optimized Amazon Linux AMI** – This AMI is based off of Amazon Linux 1. We recommend that you migrate your workloads to the Amazon ECS-optimized Amazon Linux 2 AMI. Support for the Amazon ECS-optimized Amazon Linux AMI ends no later than June 30, 2020.

• **Amazon ECS-optimized Windows 2019 AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system. For more information, see [Windows Containers (p. 610)].

• **Amazon ECS-optimized Windows 2016 AMI** – Available for launching your Amazon ECS container instances on the Windows operating system. For more information, see [Windows Containers (p. 610)].

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 ID, for each variant can be retrieved programmatically. For more information, see [Retrieving Amazon ECS-Optimized AMI Metadata (p. 91)].

View the AMI IDs on one of the following tabs, according to the variant you choose.

**Amazon Linux 2**

The current Amazon ECS-optimized Amazon Linux 2 AMI consists of:

- The latest minimal version of the Amazon Linux 2
- The latest version of the Amazon ECS container agent (1.28.0)
- The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
- The latest version of the `ecs-init` package to run and monitor the Amazon ECS agent (1.28.0-1)

The following table lists the current Amazon ECS-optimized Amazon Linux 2 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-012ca23958772e5f</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-00cf4737e2b886eb</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06d07f0156b1d44a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0a9f5be2a016ddee</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-067f4f7124e746eb</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0e52aad6ac7733a8a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-08834c857e502a560d</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-04322e867758d97</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0047bfdb16f1f67265</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0947584732e55653a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0498c464ec4d2ba75</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-096a38c97b80c386</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0dddc4daca446498</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0c5ab45f676ab6d84</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0204aa6a92a545686</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-07273195833e4f2d6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-00d85164873aaa4d</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06473be43b0f7769</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-607c0001</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

Amazon Linux 2 (arm64)

The current Amazon ECS-optimized Amazon Linux 2 (arm64) AMI consists of:

- The latest minimal version of the Amazon Linux 2
The latest version of the Amazon ECS container agent (1.28.0)
The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
The latest version of the `ecs-init` package to run and monitor the Amazon ECS agent (1.28.0-1)

The following table lists the current Amazon ECS-optimized Amazon Linux 2 (arm64) AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-arm64-eb3</td>
<td>ami-030392040d1aed9</td>
<td><img src="launch_instance" alt="Launch instance" /></td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-arm64-eb</td>
<td>ami-0e7126260e3c3f9</td>
<td><img src="launch_instance" alt="Launch instance" /></td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-arm64-eb</td>
<td>ami-037a4247c72ff57</td>
<td><img src="launch_instance" alt="Launch instance" /></td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-arm64-eb</td>
<td>ami-04d7703e789bab9</td>
<td><img src="launch_instance" alt="Launch instance" /></td>
</tr>
</tbody>
</table>

Amazon Linux 2 (GPU)

The current Amazon ECS GPU-optimized AMI consists of the following:

- The latest minimal version of the Amazon Linux 2
- The latest version of the Amazon ECS container agent (1.28.0)
- The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
- The latest version of the `ecs-init` package to run and monitor the Amazon ECS agent (1.28.0-1)
- The recommended NVIDIA driver version (418.40.04)
- The NVIDIA container runtime hook version (v1.4.0-1)
- The recommended CUDA version (9.2.88)

The following table lists the current Amazon ECS GPU-optimized AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-eb</td>
<td>ami-0f132b270b9aabe</td>
<td><img src="launch_instance" alt="Launch instance" /></td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-eb</td>
<td>ami-0b0ae551a867891</td>
<td><img src="launch_instance" alt="Launch instance" /></td>
</tr>
</tbody>
</table>
## Amazon ECS-optimized AMIs

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-west-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0ca127ab2bfadf61f</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-083612cfef21db16</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0ce92ebe7a582257</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0a90420c9bdbe47</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0aa98500408657f</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-066ec8c54f8a647f</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-03893996ba1620ba</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-01d49902dbdd248</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-00136313cb23ebda</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0eda13f6b29f5f6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0a004e87a7109c8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0123f684a725867f</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-051ff72f7e0a8ba6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0ba990e211024cb</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

API Version 2014-11-13
Amazon Elastic Container Service Developer Guide

Amazon ECS-optimized AMIs

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-west-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0bae1ac39b3ab1ceb</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06d9106a2e6079a8ab</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-01ee887a6e2d4a6f3</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-05ecc99800a097f6b5</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-bd7b07dc</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

Amazon Linux AMI

The current Amazon ECS-optimized Amazon Linux AMI consists of:

- The latest minimal version of the Amazon Linux AMI
- The latest version of the Amazon ECS container agent (1.28.0)
- The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
- The latest version of the ecs-init package to run and monitor the Amazon ECS agent (1.28.0-1)

The following table lists the current Amazon ECS-optimized Amazon Linux AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-06a8ae0e0ecd0e80e6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0750ab1027b6314b7</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-03fe84be94ca9cc4b4</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-043c4e6bff652b92d</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0d772c70a2d689eb7</td>
<td>Launch instance</td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------</td>
<td>-----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-062ef2a2561c936ed</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005600074f3aa4a1</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0a80c5ae873c08c932</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0b78efd7fafa3f9268</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0c5058003c511da666</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005ff694f167b5719</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005ce40cc5e50b58</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0066a51374a90a2f81</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-06a20f16dd2f507949</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-087de2c1b546b7a9b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-09b15689425532595</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-03ca259ae4cb868192</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-055c29a7d5fc2d4e11</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0d83f147ba8af34682</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>
Amazon Elastic Container Service Developer Guide
Amazon ECS-optimized AMIs

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-gov-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0b93f4db7ff03a1b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-e97a0688</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

Windows Server 2019

The current Amazon ECS-optimized Windows 2019 AMI consists of:

- The latest version of Microsoft Windows Server 2019
- Docker EE version 18.09.4
- Amazon ECS container agent version 1.27.0

The following table lists the current Amazon ECS-optimized Windows 2019 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0a4540e9bef884ab</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0f7cc2a4e9cb931a9</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0ca9f58a76b75a9e9</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-09a6b4fc9786621fa</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0de9f680eb13955e6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-052dc171cf22e5b2</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0fda456670ecdd2f</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0d073901cb231de8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------</td>
<td>---------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-06cef3b9805e5e6bb</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-09f37f76841876c22</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-02942c6681667846</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-066eff0f2473d2b5</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-09bfe64c8c31022b3</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-078d39ec1c8b11d0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-05da69b2d804943</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-01fbd6d84ec8b36</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0a64405322f3a0b8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0ac8048de25ce4277</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0c4c54ea7fe80d4d3</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-d91f63b8</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

**Windows Server 2016**

The current Amazon ECS-optimized Windows 2016 AMI consists of:

- The latest version of Microsoft Windows Server 2016
- Docker EE version 18.03.1-ee-7
- Amazon ECS container agent version 1.26.0
The following table lists the current Amazon ECS-optimized Windows 2016 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>Windows_Server-2016-</td>
<td>ami-00c56e74f090d6f</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>us-east-1</td>
<td>Windows_Server-2016-</td>
<td>ami-0ed2f29599018e7</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>us-west-1</td>
<td>Windows_Server-2016-</td>
<td>ami-01b55f7fe967f73</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>us-west-2</td>
<td>Windows_Server-2016-</td>
<td>ami-07f6b98dc6c8067</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>Windows_Server-2016-</td>
<td>ami-07839df9eece55acb</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>Windows_Server-2016-</td>
<td>ami-071b78467d935544</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ap-south-1</td>
<td>Windows_Server-2016-</td>
<td>ami-0183732d8e0fd56</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>Windows_Server-2016-</td>
<td>ami-0a6c13d83c0fdbe5</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>Windows_Server-2016-</td>
<td>ami-06d33f81ca838455</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ca-central-1</td>
<td>Windows_Server-2016-</td>
<td>ami-098ad73a3005b656</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cn-north-1</td>
<td>Windows_Server-2016-</td>
<td>ami-0b484446add9a27d</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>Windows_Server-2016-</td>
<td>ami-047cc7df873d123</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eu-central-1</td>
<td>Windows_Server-2016-</td>
<td>ami-0f7386282aa13af5</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eu-north-1</td>
<td>Windows_Server-2016-</td>
<td>ami-06e3cb4d2875b17</td>
<td>Launch instance</td>
</tr>
<tr>
<td></td>
<td>English-Full-ECS_Optimized-2019.03.07</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to Launch the Latest Amazon ECS-Optimized Amazon Linux AMI

The following are several ways that you can launch the latest Amazon ECS-optimized AMI into your cluster:

- The Amazon ECS console first-run wizard launches your container instances with the latest Amazon ECS-optimized AMI. For more information, see Getting Started with Amazon ECS (p. 19).
- You can launch your container instances manually in the Amazon EC2 console by following the procedures in Launching an Amazon ECS Container Instance (p. 101). You could also choose the EC2 Console Link in one of the tables below that corresponds to your cluster's region.
- You can retrieve the current Amazon ECS-optimized AMI programmatically using the SSM API. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 91).
- Use the current Amazon ECS-optimized AMI ID, according to the variant you choose, below to launch your instance using the AWS CLI, the AWS SDKs, or an AWS CloudFormation template.

Amazon Linux 2

The current Amazon ECS-optimized Amazon Linux 2 AMI consists of:

- The latest minimal version of the Amazon Linux 2
- The latest version of the Amazon ECS container agent (1.28.0)
- The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
- The latest version of the `ecs-init` package to run and monitor the Amazon ECS agent (1.28.0-1)

The following table lists the current Amazon ECS-optimized Amazon Linux 2 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-012ca23958772cf3</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-00cf4737e238866</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06d07f0156b1d44</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0a9f5be2a016dcb3</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-067f4f7124e746e</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0e52aad6ac7733a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-08834c8c57e502d</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-04322e867758d97</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0047bdfb16f677b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0947584732e5565</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0498c464ec4d2bad</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-096a38c97b80cd8</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>
### Amazon Elastic Container Service Developer Guide

#### How to Launch the Latest Amazon ECS-Optimized Amazon Linux AMI

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-north-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0dddc4daca44e6eb</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0c5abd4f676ab</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0204aa92a5456</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0727319583e4f2</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-00d851648873aa</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06473be43b0f776</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-607c0001</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

**Amazon Linux 2 (arm64)**

The current Amazon ECS-optimized Amazon Linux 2 (arm64) AMI consists of:

- The latest minimal version of the Amazon Linux 2
- The latest version of the Amazon ECS container agent (1.28.0)
- The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
- The latest version of the `ecs-init` package to run and monitor the Amazon ECS agent (1.28.0-1)

The following table lists the current Amazon ECS-optimized Amazon Linux 2 (arm64) AMI IDs by Region.
### Amazon Linux 2 (GPU)

The current Amazon ECS GPU-optimized AMI consists of the following:

- The latest minimal version of the Amazon Linux 2
- The latest version of the Amazon ECS container agent (1.28.0)
- The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
- The latest version of the *ecs-init* package to run and monitor the Amazon ECS agent (1.28.0-1)
- The recommended NVIDIA driver version (418.40.04)
- The NVIDIA container runtime hook version (v1.4.0-1)
- The recommended CUDA version (9.2.88)

The following table lists the current Amazon ECS GPU-optimized AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-arm64-ebs</td>
<td>ami-04d7703e789bab4a</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>us-east-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0f132b270b9abeb4a</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0b0ae551a867891da</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>us-west-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0ca127ab2baf6e5f</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-083612cfef21db1c</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0ce92eb7a582255a</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0a090420c9bd60e4</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0aa98500408657f</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06edc88c54f8a637</td>
<td><img src="https://console.aws.amazon.com/ec2/home" alt="Launch instance" /></td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------</td>
<td>------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-03893996ba1620b5b8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-01d49902dbdd2483</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-00136313cb23e1b60</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0eda13fcf629f5f2e0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0a004e87a7189c0b8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0123f684a7258c77e0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-051ff72feco0a8bae0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0ba90e211024eb2e0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0bae139b3ab1e2b2e0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06d9106a2e607a8b2e0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-01leece87a06c24eb2e0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-05d93990a097b4eb2e0</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>amzn2-ami-ecs-gpu-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-bd7b07dc</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

Amazon Linux AMI

The current Amazon ECS-optimized Amazon Linux AMI consists of:

- The latest minimal version of the Amazon Linux AMI
• The latest version of the Amazon ECS container agent (1.28.0)
• The recommended version of Docker for the latest Amazon ECS container agent (18.06.1-ce)
• The latest version of the `ecs-init` package to run and monitor the Amazon ECS agent (1.28.0-1)

The following table lists the current Amazon ECS-optimized Amazon Linux AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-06a8ae0ecd30e80</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0750ab1027b6314</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-03fe84be94ca9cc</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-043c4e6b6f652b9</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0d77c702a2d689e</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-062ef2a2561c936</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-005600074f3a444</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0a80c5ae873c08c</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0b78fd7fae3f9b5</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0c5058003c511da</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005ff69f167b55</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005ce40ceef58b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>---------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0066a51374afa1beb</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-06a20f16dd2f507</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-087de2c1b54c6b2</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-09b156894255325</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-03ca259ae4cb86f5e7</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-055c29a7d5fc2d8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0d83f147ba8afa39</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0b93f4db7ff03a13</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-e97a0688</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

Windows Server 2019

The current Amazon ECS-optimized Windows 2019 AMI consists of:

- The latest version of Microsoft Windows Server 2019
- Docker EE version 18.09.4
- Amazon ECS container agent version 1.27.0

The following table lists the current Amazon ECS-optimized Windows 2019 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0a4548e9bef884a58</td>
<td>Launch instance</td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------</td>
<td>----------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>us-east-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0f7cc2a4e9cb931b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0caa9f58a76b75db</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-09a64fc9786621ba</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0de9680eb139f5cb</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-052dc171cf22efb2</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0fda456670eddaa1</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0d073901cb231d48</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-06cef3b9805e5eb3</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-09f3776841876c2a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-02942c6681667844</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-066ef0f2473d2ea7</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-09bff64c8c31022a5</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-078d39ec1c8b11db4</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-05da69b2d804943b</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>
### How to Launch the Latest Amazon ECS-Optimized Amazon Linux AMI

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-west-2</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-01fbd6d84ec8b3678</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0a6440532f93a0662</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0ac8048de25ce428a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-0c4c54ea7fe80d47a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10</td>
<td>ami-d91f63b8</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

**Windows Server 2016**

The current Amazon ECS-optimized Windows 2016 AMI consists of:

- The latest version of Microsoft Windows Server 2016
- Docker EE version 18.03.1-ee-7
- Amazon ECS container agent version 1.26.0

The following table lists the current Amazon ECS-optimized Windows 2016 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-00c56e74f090d6f76</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-0ed2f295990187e72</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-01b55f7fe967f721b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-07f6b98dc6c8067e2</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-07839df9eac55ac8a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------</td>
<td>-----------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-071b78467d9d355</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-0183732d8e0fd56</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-0a6c13d83c0fd9b6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-06d33f81ca83845</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-098ad73a3005b665</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-0b48446add927a1</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-047cc7df873d123</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-0f7386282aa13a08</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-06e3cb4d2875bb76</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-04a2fa8ce0fc20ca</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-0fac4f3bdab9cc6a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-06a5b6fc5225119</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-05889298c47e6d5</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-0b6f703732ae49de</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>
Amazon Elastic Container Service Developer Guide
AMI Versions

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-gov-west-1</td>
<td>Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07</td>
<td>ami-9d91fafe</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>

For more information about previous versions and the corresponding Docker and Amazon ECS container agent versions, see Amazon ECS-Optimized Amazon Linux AMI Container Agent Versions (p. 144).

Amazon ECS-optimized AMI Versions

This topic lists 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 Retrieving Amazon ECS-Optimized AMI Metadata (p. 91).

Topics
- Amazon ECS-Optimized Amazon Linux 2 AMI Versions (p. 82)
- Amazon ECS-optimized Amazon Linux 2 (arm64) AMI Versions (p. 83)
- Amazon ECS GPU-optimized AMI Versions (p. 83)
- Amazon ECS-optimized Amazon Linux AMI Versions (p. 84)
- Amazon ECS-optimized Windows 2019 AMI Versions (p. 86)
- Amazon ECS-optimized Windows 2016 AMI Versions (p. 86)

Amazon ECS-Optimized Amazon Linux 2 AMI Versions

The table below lists the current and previous versions of the Amazon ECS-optimized Amazon Linux 2 AMI and their corresponding versions of the Amazon ECS container agent, Docker, and the ecs-init package.

<table>
<thead>
<tr>
<th>Amazon ECS-optimized Amazon Linux 2 AMI</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
<th>ecs-init version</th>
</tr>
</thead>
<tbody>
<tr>
<td>20190402</td>
<td>1.27.0</td>
<td>18.06.1-ce</td>
<td>1.27.0-1</td>
</tr>
<tr>
<td>20190301</td>
<td>1.26.0</td>
<td>18.06.1-ce</td>
<td>1.26.0-1</td>
</tr>
<tr>
<td>20190215</td>
<td>1.25.3</td>
<td>18.06.1-ce</td>
<td>1.25.3-1</td>
</tr>
<tr>
<td>20190204</td>
<td>1.25.2</td>
<td>18.06.1-ce</td>
<td>1.25.2-1</td>
</tr>
<tr>
<td>20190127</td>
<td>1.25.1</td>
<td>18.06.1-ce</td>
<td>1.25.1-1</td>
</tr>
<tr>
<td>20190118</td>
<td>1.25.0</td>
<td>18.06.1-ce</td>
<td>1.25.0-1</td>
</tr>
<tr>
<td>20190107</td>
<td>1.24.0</td>
<td>18.06.1-ce</td>
<td>1.24.0-1</td>
</tr>
<tr>
<td>20181112</td>
<td>1.22.0</td>
<td>18.06.1-ce</td>
<td>1.22.0-1</td>
</tr>
<tr>
<td>20181016</td>
<td>1.20.3</td>
<td>18.06.1-ce</td>
<td>1.21.0-1</td>
</tr>
</tbody>
</table>
The current Amazon ECS-optimized Amazon Linux 2 AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/recommended
```

### Amazon ECS-optimized Amazon Linux 2 (arm64) AMI Versions

The table below lists the current and previous versions of the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package.

<table>
<thead>
<tr>
<th>Amazon ECS-optimized Amazon Linux 2 (arm64) AMI</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
<th><code>ecs-init</code> version</th>
</tr>
</thead>
<tbody>
<tr>
<td>20190403</td>
<td>1.27.0</td>
<td>18.06.1-ce</td>
<td>1.27.0-1</td>
</tr>
<tr>
<td>20190301</td>
<td>1.26.0</td>
<td>18.06.1-ce</td>
<td>1.26.0-1</td>
</tr>
<tr>
<td>20190215</td>
<td>1.25.3</td>
<td>18.06.1-ce</td>
<td>1.25.3-1</td>
</tr>
<tr>
<td>20190204</td>
<td>1.25.2</td>
<td>18.06.1-ce</td>
<td>1.25.2-1</td>
</tr>
<tr>
<td>20190127</td>
<td>1.25.1</td>
<td>18.06.1-ce</td>
<td>1.25.1-1</td>
</tr>
<tr>
<td>20190119</td>
<td>1.25.0</td>
<td>18.06.1-ce</td>
<td>1.25.0-1</td>
</tr>
<tr>
<td>20181120</td>
<td>1.22.0</td>
<td>18.06.1-ce</td>
<td>1.22.0-1</td>
</tr>
</tbody>
</table>

The current Amazon ECS-optimized Amazon Linux 2 (arm64) AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/arm64/recommended
```

### Amazon ECS GPU-optimized AMI Versions

The table below lists the current and previous versions of the Amazon ECS GPU-optimized AMI and their corresponding versions of the Amazon ECS container agent, Docker, `ecs-init` package, and NVIDIA driver.

<table>
<thead>
<tr>
<th>Amazon ECS GPU-optimized AMI</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
<th><code>ecs-init</code> version</th>
<th>NVIDIA driver version</th>
</tr>
</thead>
<tbody>
<tr>
<td>20190402</td>
<td>1.27.0</td>
<td>18.06.1-ce</td>
<td>1.27.0-1</td>
<td>418.40.04</td>
</tr>
<tr>
<td>20190321</td>
<td>1.26.0</td>
<td>18.06.1-ce</td>
<td>1.26.0-1</td>
<td>410.104</td>
</tr>
<tr>
<td>20190301</td>
<td>1.26.0</td>
<td>18.06.1-ce</td>
<td>1.26.0-1</td>
<td>396.26</td>
</tr>
<tr>
<td>20190215</td>
<td>1.25.3</td>
<td>18.06.1-ce</td>
<td>1.25.3-1</td>
<td>396.26</td>
</tr>
<tr>
<td>20190204</td>
<td>1.25.2</td>
<td>18.06.1-ce</td>
<td>1.25.2-1</td>
<td>396.26</td>
</tr>
<tr>
<td>20190127</td>
<td>1.25.1</td>
<td>18.06.1-ce</td>
<td>1.25.1-1</td>
<td>396.26</td>
</tr>
</tbody>
</table>
You can retrieve the current Amazon ECS GPU-optimized AMI using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended
```

### Amazon ECS-optimized Amazon Linux AMI Versions

The table below lists the current and previous versions of the Amazon ECS-optimized Amazon Linux AMI and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package.

<table>
<thead>
<tr>
<th>Amazon ECS-optimized Amazon Linux AMI</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
<th>ecs-init version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.03.p</td>
<td>1.27.0</td>
<td>18.06.1-ce</td>
<td>1.27.0-1</td>
</tr>
<tr>
<td>2018.03.o</td>
<td>1.26.0</td>
<td>18.06.1-ce</td>
<td>1.26.0-1</td>
</tr>
<tr>
<td>2018.03.n</td>
<td>1.25.3</td>
<td>18.06.1-ce</td>
<td>1.25.3-1</td>
</tr>
<tr>
<td>2018.03.m</td>
<td>1.25.2</td>
<td>18.06.1-ce</td>
<td>1.25.2-1</td>
</tr>
<tr>
<td>2018.03.l</td>
<td>1.25.1</td>
<td>18.06.1-ce</td>
<td>1.25.1-1</td>
</tr>
<tr>
<td>2018.03.k</td>
<td>1.25.0</td>
<td>18.06.1-ce</td>
<td>1.25.0-1</td>
</tr>
<tr>
<td>2018.03.j</td>
<td>1.24.0</td>
<td>18.06.1-ce</td>
<td>1.24.0-1</td>
</tr>
<tr>
<td>2018.03.i</td>
<td>1.22.0</td>
<td>18.06.1-ce</td>
<td>1.22.0-1</td>
</tr>
<tr>
<td>2018.03.h</td>
<td>1.21.0</td>
<td>18.06.1-ce</td>
<td>1.21.0-1</td>
</tr>
<tr>
<td>2018.03.g</td>
<td>1.20.3</td>
<td>18.06.1-ce</td>
<td>1.20.3-1</td>
</tr>
<tr>
<td>2018.03.f</td>
<td>1.20.2</td>
<td>18.06.1-ce</td>
<td>1.20.2-1</td>
</tr>
<tr>
<td>2018.03.e</td>
<td>1.20.1</td>
<td>18.03.1-ce</td>
<td>1.20.1-1</td>
</tr>
<tr>
<td>2018.03.d</td>
<td>1.20.0</td>
<td>18.03.1-ce</td>
<td>1.20.0-1</td>
</tr>
<tr>
<td>2018.03.c</td>
<td>1.19.1</td>
<td>18.03.1-ce</td>
<td>1.19.1-1</td>
</tr>
<tr>
<td>2018.03.b</td>
<td>1.19.0</td>
<td>18.03.1-ce</td>
<td>1.19.0-1</td>
</tr>
<tr>
<td>2018.03.a</td>
<td>1.18.0</td>
<td>17.12.1-ce</td>
<td>1.18.0-1</td>
</tr>
<tr>
<td>2017.09.l</td>
<td>1.17.3</td>
<td>17.12.1-ce</td>
<td>1.17.3-1</td>
</tr>
<tr>
<td>2017.09.k</td>
<td>1.17.2</td>
<td>17.12.0-ce</td>
<td>1.17.2-1</td>
</tr>
<tr>
<td>2017.09.j</td>
<td>1.17.2</td>
<td>17.12.0-ce</td>
<td>1.17.2-1</td>
</tr>
<tr>
<td>Amazon ECS-optimized Amazon Linux AMI</td>
<td>Amazon ECS container agent version</td>
<td>Docker version</td>
<td>ecs-init version</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------</td>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>2017.09.i</td>
<td>1.17.1</td>
<td>17.09.1-ce</td>
<td>1.17.1-1</td>
</tr>
<tr>
<td>2017.09.h</td>
<td>1.17.0</td>
<td>17.09.1-ce</td>
<td>1.17.0-2</td>
</tr>
<tr>
<td>2017.09.g</td>
<td>1.16.2</td>
<td>17.09.1-ce</td>
<td>1.16.2-1</td>
</tr>
<tr>
<td>2017.09.f</td>
<td>1.16.1</td>
<td>17.06.2-ce</td>
<td>1.16.1-1</td>
</tr>
<tr>
<td>2017.09.e</td>
<td>1.16.1</td>
<td>17.06.2-ce</td>
<td>1.16.1-1</td>
</tr>
<tr>
<td>2017.09.d</td>
<td>1.16.0</td>
<td>17.06.2-ce</td>
<td>1.16.0-1</td>
</tr>
<tr>
<td>2017.09.c</td>
<td>1.15.2</td>
<td>17.06.2-ce</td>
<td>1.15.1-1</td>
</tr>
<tr>
<td>2017.09.b</td>
<td>1.15.1</td>
<td>17.06.2-ce</td>
<td>1.15.1-1</td>
</tr>
<tr>
<td>2017.09.a</td>
<td>1.15.0</td>
<td>17.06.2-ce</td>
<td>1.15.0-4</td>
</tr>
<tr>
<td>2017.03.g</td>
<td>1.14.5</td>
<td>17.03.2-ce</td>
<td>1.14.5-1</td>
</tr>
<tr>
<td>2017.03.f</td>
<td>1.14.4</td>
<td>17.03.2-ce</td>
<td>1.14.4-1</td>
</tr>
<tr>
<td>2017.03.e</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2017.03.d</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2017.03.c</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2017.03.b</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2016.09.g</td>
<td>1.14.1</td>
<td>1.12.6</td>
<td>1.14.1-1</td>
</tr>
<tr>
<td>2016.09.e</td>
<td>1.14.0</td>
<td>1.12.6</td>
<td>1.14.0-1</td>
</tr>
<tr>
<td>2016.09.c</td>
<td>1.13.1</td>
<td>1.11.2</td>
<td>1.13.1-1</td>
</tr>
<tr>
<td>2016.09.b</td>
<td>1.13.1</td>
<td>1.11.2</td>
<td>1.13.1-1</td>
</tr>
<tr>
<td>2016.09.a</td>
<td>1.13.0</td>
<td>1.11.2</td>
<td>1.13.0-1</td>
</tr>
<tr>
<td>2016.03.j</td>
<td>1.13.0</td>
<td>1.11.2</td>
<td>1.13.0-1</td>
</tr>
<tr>
<td>2016.03.i</td>
<td>1.12.2</td>
<td>1.11.2</td>
<td>1.12.2-1</td>
</tr>
<tr>
<td>2016.03.h</td>
<td>1.12.1</td>
<td>1.11.2</td>
<td>1.12.1-1</td>
</tr>
<tr>
<td>2016.03.g</td>
<td>1.12.0</td>
<td>1.11.2</td>
<td>1.12.0-1</td>
</tr>
<tr>
<td>2016.03.f</td>
<td>1.11.1</td>
<td>1.11.2</td>
<td>1.11.1-1</td>
</tr>
<tr>
<td>2016.03.e</td>
<td>1.11.0</td>
<td>1.11.2</td>
<td>1.11.0-1</td>
</tr>
<tr>
<td>2016.03.d</td>
<td>1.10.0</td>
<td>1.11.1</td>
<td>1.10.0-1</td>
</tr>
<tr>
<td>2016.03.c</td>
<td>1.10.0</td>
<td>1.11.1</td>
<td>1.10.0-1</td>
</tr>
</tbody>
</table>
### Amazon ECS-optimized Amazon Linux AMI

<table>
<thead>
<tr>
<th>AMI Version</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
<th>ecs-init version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016.03.b</td>
<td>1.9.0</td>
<td>1.9.1</td>
<td>1.9.0-1</td>
</tr>
<tr>
<td>2016.03.a</td>
<td>1.8.2</td>
<td>1.9.1</td>
<td>1.8.2-1</td>
</tr>
<tr>
<td>2015.09.g</td>
<td>1.8.1</td>
<td>1.9.1</td>
<td>1.8.1-1</td>
</tr>
<tr>
<td>2015.09.f</td>
<td>1.8.0</td>
<td>1.9.1</td>
<td>1.8.0-1</td>
</tr>
<tr>
<td>2015.09.e</td>
<td>1.7.1</td>
<td>1.9.1</td>
<td>1.7.1-1</td>
</tr>
<tr>
<td>2015.09.d</td>
<td>1.7.1</td>
<td>1.9.1</td>
<td>1.7.1-1</td>
</tr>
<tr>
<td>2015.09.c</td>
<td>1.7.0</td>
<td>1.7.1</td>
<td>1.7.0-1</td>
</tr>
<tr>
<td>2015.09.b</td>
<td>1.6.0</td>
<td>1.7.1</td>
<td>1.6.0-1</td>
</tr>
<tr>
<td>2015.09.a</td>
<td>1.5.0</td>
<td>1.7.1</td>
<td>1.5.0-1</td>
</tr>
<tr>
<td>2015.03.g</td>
<td>1.4.0</td>
<td>1.7.1</td>
<td>1.4.0-2</td>
</tr>
<tr>
<td>2015.03.f</td>
<td>1.4.0</td>
<td>1.6.2</td>
<td>1.4.0-1</td>
</tr>
<tr>
<td>2015.03.e</td>
<td>1.3.1</td>
<td>1.6.2</td>
<td>1.3.1-1</td>
</tr>
<tr>
<td>2015.03.d</td>
<td>1.2.1</td>
<td>1.6.2</td>
<td>1.2.0-2</td>
</tr>
<tr>
<td>2015.03.c</td>
<td>1.2.0</td>
<td>1.6.2</td>
<td>1.2.0-1</td>
</tr>
<tr>
<td>2015.03.b</td>
<td>1.1.0</td>
<td>1.6.0</td>
<td>1.0-3</td>
</tr>
<tr>
<td>2015.03.a</td>
<td>1.0.0</td>
<td>1.5.0</td>
<td>1.0-1</td>
</tr>
</tbody>
</table>

### Amazon ECS-optimized Windows 2019 AMI Versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows 2019 AMI and their corresponding versions of the Amazon ECS container agent and Docker.

<table>
<thead>
<tr>
<th>AMI Version</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.05.10</td>
<td>1.27.0</td>
<td>18.09.4</td>
</tr>
</tbody>
</table>

### Amazon ECS-optimized Windows 2016 AMI Versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows 2016 AMI and their corresponding versions of the Amazon ECS container agent and Docker.

<table>
<thead>
<tr>
<th>AMI Version</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.03.07</td>
<td>1.26.0</td>
<td>18.03.1-ee-7</td>
</tr>
</tbody>
</table>
AMI Storage Configuration

The following describes the storage configuration for each of the Amazon ECS-optimized AMIs.

Topics

- Amazon Linux 2 Storage Configuration (p. 87)
- Amazon ECS-optimized Amazon Linux AMI Storage Configuration (p. 87)

Amazon Linux 2 Storage Configuration

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 ext4, and Docker uses the overlay2 storage driver. For more information, see Use the OverlayFS storage driver in the Docker documentation.

Amazon ECS-optimized Amazon Linux AMI Storage Configuration

By default, the Amazon ECS-optimized Amazon Linux AMI ships with 30 GiB of total storage. You can modify this value at launch time to increase or decrease the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata. The sections below describe the storage configuration of the Amazon ECS-optimized Amazon Linux AMI, based on the AMI version.

Version 2015.09.d and Later

Amazon ECS-optimized Amazon Linux AMIs from version 2015.09.d and later launch with an 8-GiB volume for the operating system that is attached at /dev/xvda and mounted as the root of the file system. There is an additional 22-GiB volume that is attached at /dev/xvdcz that Docker uses for image and metadata storage. The volume is configured as a Logical Volume Management (LVM) device and it is accessed directly by Docker via the devicemapper backend. Because the volume is not mounted, you cannot use standard storage information commands (such as df -h) to determine the available storage. However, you can use LVM commands and docker info to find the available storage by following the procedure below. For more information, see the LVM HOWTO in The Linux Documentation Project.

Note
You can increase these default volume sizes by changing the block device mapping settings for your instances when you launch them; however, you cannot specify a smaller volume size than the default. For more information, see Block Device Mapping in the Amazon EC2 User Guide for Linux Instances.

The docker-storage-setup utility configures the LVM volume group and logical volume for Docker when the instance launches. By default, docker-storage-setup creates a volume group called docker, adds /dev/xvdcz as a physical volume to that group. It then creates a logical volume called docker-pool that uses 99% of the available storage in the volume group. The remaining 1% of the available storage is reserved for metadata.

Note
Earlier Amazon ECS-optimized Amazon Linux AMI versions (2015.09.d to 2016.03.a) create a logical volume that uses 40% of the available storage in the volume group. When the logical volume becomes 60% full, the logical volume is increased in size by 20%.
To determine the available storage for Docker

- You can use the LVM commands, `vgs` and `lvs`, or the `docker info` command to view available storage for Docker.

  **Note**
  The LVM command output displays storage values in GiB (2^30 bytes), and `docker info` displays storage values in GB (10^9 bytes).

  a. You can view the available storage in the volume group with the `vgs` command. This command shows the total size of the volume group and the available space in the volume group that can be used to grow the logical volume. The example below shows a 22-GiB volume with 204 MiB of free space.

    ```
    [ec2-user ~]$ sudo vgs
    Output:
    VG   #PV #LV #SN Attr  VSize  VFree
    docker 1 1 0 wz--n- 22.00g 204.00m
    ```

  b. You can view the available space in the logical volume with the `lvs` command. The example below shows a logical volume that is 21.75 GiB in size, and it is 7.63% full. This logical volume can grow until there is no more free space in the volume group.

    ```
    [ec2-user@ ~]$ sudo lvs
    Output:
    LV      VG  Attr   LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync
    Convert
    docker-pool docker twi-aot--- 21.75g             7.63   4.96
    ```

  c. The `docker info` command also provides information about how much data space it is using, and how much data space is available. However, its available space value is based on the logical volume size that it is using.

    **Note**
    Because `docker info` displays storage values as GB (10^9 bytes), instead of GiB (2^30 bytes), the values displayed here look larger for the same amount of storage displayed with `lvs`. However, the values are equal (23.35 GB = 21.75 GiB).

    ```
    [ec2-user ~]$ docker info | grep "Data Space"
    Output:
    Data Space Used: 1.782 GB
    Data Space Total: 23.35 GB
    Data Space Available: 21.57 GB
    ```

To extend the Docker logical volume

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 are unable to do this, you can add storage to the volume group that Docker uses and extend its logical volume by following these steps.
Note

If your container instance storage is filling up too quickly, there are a few actions that you can take to reduce this effect:

- (Amazon ECS container agent 1.8.0 and later) 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 may 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 (p. 152).

- 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 cannot be inspected later, and deleted images must be pulled again before starting new containers from them.

  To remove non-running containers, execute the following command on your container instance:

  ```
  # docker rm $(docker ps -aq)
  ```

  To remove unused images, execute the following command on your container instance:

  ```
  # docker rmi $(docker images -q)
  ```

- 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/"
  ```

1. Create a new Amazon EBS volume in the same Availability Zone as your container instance. For more information, see Creating an Amazon EBS Volume in the Amazon EC2 User Guide for Linux Instances.

2. Attach the volume to your container instance. The default location for the Docker data volume is `/dev/xvdcz`. For consistency, attach additional volumes in reverse alphabetical order from that device name (for example, `/dev/xvdcy`). For more information, see Attaching an Amazon EBS Volume to an Instance in the Amazon EC2 User Guide for Linux Instances.

3. Connect to your container instance using SSH. For more information, see Connect to Your Container Instance (p. 113).

4. Check the size of your `docker-pool` logical volume. The example below shows a logical volume of 409.19 GiB.

   ```
   [ec2-user ~]# sudo lvs
   ```

   Output:

<table>
<thead>
<tr>
<th>LV</th>
<th>VG</th>
<th>Attr</th>
<th>LSize</th>
<th>Pool Origin</th>
<th>Data%</th>
<th>Meta%</th>
<th>Move</th>
<th>Log</th>
<th>Cpy%</th>
<th>Sync</th>
<th>Convert</th>
</tr>
</thead>
<tbody>
<tr>
<td>docker-pool</td>
<td>docker twi-aot---</td>
<td>409.19g</td>
<td>0.16</td>
<td>0.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Check the current available space in your volume group. The example below shows 612.75 GiB in the VFree column.
Add the new volume to the `docker` volume group, substituting the device name to which you attached the new volume. In this example, a 1-TiB volume was previously added and attached to `/dev/xvdcy`.

```
[ec2-user ~]$
sudo vgextend docker /dev/xvdcy
```

Physical volume "/dev/sdcy" successfully created
Volume group "docker" successfully extended

7. Verify that your volume group size has increased with the `vgs` command. The `VFree` column should show the increased storage size. The example below now has 1.6 TiB in the `VFree` column, which is 1 TiB larger than it was previously. Your `VFree` column should be the sum of the original `VFree` value and the size of the volume you attached.

```
[ec2-user ~]$
sudo vgs
```

```
Output:
VG     #PV #LV #SN Attr   VSize  VFree
docker  2   1   0 wz--n- 2.00t  1.60t
```

8. Extend the `docker-pool` logical volume with the size of the volume you added earlier. The command below adds 1024 GiB to the logical volume, which is entered as `1024G`.

```
[ec2-user ~]$
sudo lvextend -L+ 1024G /dev/docker/docker-pool
```

Output:
```
Size of logical volume docker/docker-pool_tdata changed from 409.19 GiB (104752 extents) to 1.40 TiB (366896 extents).
Logical volume docker-pool successfully resized
```

9. Verify that your logical volume has increased in size.

```
[ec2-user ~]$
sudo lvs
```

Output:
```
LV          VG     Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
docker-pool docker twi-aot--- 1.40t
```

10. (Optional) Verify that `docker info` also recognizes the added storage space.

**Note**
Because `docker info` displays storage values as GB (10^9 bytes), instead of GiB (2^30 bytes), the values displayed here look larger for the same amount of storage displayed with `lvs`. However, the values are equal (1.539 TB = 1.40 TiB).
Version 2015.09.c and Earlier

Amazon ECS-optimized Amazon Linux AMIs from version 2015.09.c and earlier launch with a single 30-GiB volume that is attached at /dev/xvda and mounted as the root of the file system. This volume shares the operating system and all Docker images and metadata. You can determine the available storage on your container instance with standard storage information commands (such as `df -h`).

There is no practical way to add storage (that Docker can use) to instances launched from these AMIs without stopping them. If you find that your container instances need more storage than the default 30 GiB, you should terminate each instance. Then, launch another in its place with the latest Amazon ECS-optimized Amazon Linux AMI and a large enough data storage volume.

Retrieving Amazon ECS-Optimized AMI Metadata

The AMI ID, image name, operating system, container agent version, and runtime version for the different Amazon ECS-optimized AMIs can be programmatically retrieved by querying the SSM Parameter Store API. For more information about the SSM Parameter Store API, see `GetParameters` and `GetParametersByPath`.

**Note**

Your user account 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`

The following is the format of the parameter name.

- Amazon ECS-optimized Amazon Linux 2 AMI metadata:
  
  `/aws/service/ecs/optimized-ami/amazon-linux-2/<version>`

- Amazon ECS-optimized Amazon Linux 2 (arm64) AMI metadata:
  
  `/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/<version>`

- Amazon ECS GPU-optimized AMI metadata:
  
  `/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/<version>`

- Amazon ECS-optimized Amazon Linux AMI metadata:
  
  `/aws/service/ecs/optimized-ami/amazon-linux-2/<version>`
REtrieving Amazon ECS-Optimized AMI Metadata

The following parameter name format retrieves the metadata of the latest stable Amazon ECS-optimized Amazon Linux 2 AMI by using recommended.

/aws/service/ecs/optimized-ami/amazon-linux-2/recommended

The following is an example of the JSON object that is returned for the parameter value.

```
{
  "schema_version": 1,
  "image_name": "amzn2-ami-ecs-hvm-2.0.20181017-x86_64-ebs",
  "image_id": "ami-04a6f062c609f55b",
  "os": "Amazon Linux 2",
  "ecs_runtime_version": "Docker version 18.06.1-ce",
  "ecs_agent_version": "1.21.0"
}
```

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

The following parameter name format retrieves the image ID of the latest stable 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

- Amazon ECS-optimized Amazon Linux 2 (arm64) AMI metadata:
Retrieving Amazon ECS-Optimized AMI Metadata

- Amazon ECS-optimized Amazon Linux AMI metadata:
  
  /aws/service/ecs/optimized-ami/amazon-linux-2/recommended

- Amazon ECS-optimized Windows 2019 AMI metadata:

  /aws/service/ecs/optimized-ami/windows_server/2019/english/full/2019.05.10

- Amazon ECS-optimized Windows 2016 AMI metadata:

  /aws/service/ecs/optimized-ami/windows_server/2016/english/full/2019.03.07

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 `Windows_Server-2016-English-Full-ECS_Optimized-2018.03.26` (Windows) and later can be retrieved. For more information, see Amazon ECS-optimized AMI Versions (p. 82).

Example 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 command.

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
```

Output:

```
{
    "Parameters": [
        {
            "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/recommended",
            "Type": "String",
            "Value": "{"schema_version":1,"image_name":":amzn2-ami-ecs-hvm-2.0.20181017-x86_64-ebs","image_id":":ami-0a6be20ed8e1f055","os":":Amazon Linux 2","ecs_runtime_version":":Docker version 18.06.1-ce","ecs_agent_version":":1.21.0"},",
            "Version": 1,
            "LastModifiedDate": 1539908415.817,
            "ARN": "arn:aws:ssm:us-east-1:parameter/aws/service/ecs/optimized-ami/amazon-linux-2/recommended"
        },
        {
            "InvalidParameters": []
        }
    ]
}
```

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
```

API Version 2014-11-13

93
Output:

```json
{
   "Parameters": [
       {
           "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/recommended",
           "Type": "String",
           "Value": 
{"schema_version":1,"image_name":"amzn2-ami-ecs-hvm-2.0.20181120-arm64-ebi","image_id":"ami-053b2a8c2f3e87926","os":"Amazon Linux 2","ecs_runtime_version":"Docker version 18.06.1-ce","ecs_agent_version":"1.22.0"},
           "Version": 1,
           "LastModifiedDate": 1542745522.454,
           "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/recommended"
       }
   ],
   "InvalidParameters": []
}
```

- For the Amazon ECS GPU-optimized AMIs:

```bash
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended --region us-east-1
```

Output:

```json
{
   "Parameters": [
       {
           "Name": "/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended",
           "Type": "String",
           "Value": 
{"schema_version":1,"image_name":"amzn2-ami-ecs-gpu-hvm-2.0.20190118-x86_64-ebi","image_id":"ami-0f8776282f835efc5","os":"Amazon Linux 2","ecs_runtime_version":"Docker version 18.06.1-ce","ecs_agent_version":"1.25.0"},
           "Version": 1,
           "LastModifiedDate": 1548369045.401,
           "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/recommended"
       }
   ],
   "InvalidParameters": []
}
```

- For the Amazon ECS-optimized Amazon Linux AMIs:

```bash
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux/recommended --region us-east-1
```

Output:

```json
{
   "Parameters": [
       {
           "Name": "/aws/service/ecs/optimized-ami/amazon-linux/recommended",
           "Type": "String",
           "Value": 
{"schema_version":1,"image_name":"amzn-ami-2018.03.h-amazon-ecs-optimized","image_id":"ami-07eb698c660402d2","os":"Amazon Linux","ecs_runtime_version":"Docker version 18.06.1-ce","ecs_agent_version":"1.21.0"},
           "Version": 1,
           "LastModifiedDate": 1536118203.011,
           "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ecs/optimized-ami/amazon-linux/recommended"
       }
   ],
   "InvalidParameters": []
}
```
For the Amazon ECS-optimized Windows 2019 AMI:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended --region us-east-1
```

Output:

```
{
  "Parameters": [
    {
      "Name": "/aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended",
      "Type": "String",
      "Value": "{"schema_version":1,"image_name":"Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10","image_id":"ami-0a4548e9bef84a63","os":"Windows_Server-2019-English-Full","ecs_runtime_version":"Docker version 18.09.4","ecs_agent_version":"1.27.0"}",
      "Version": 1,
      "LastModifiedDate": 1557785029.622,
      "ARN": "arn:aws:ssm:us-east-2::parameter/aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended"
    },
    "InvalidParameters": []
  ]
}
```

For the Amazon ECS-optimized Windows 2016 AMI:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/windows_server/2016/english/full/recommended --region us-east-1
```

Output:

```
{
  "Parameters": [
    {
      "Name": "/aws/service/ecs/optimized-ami/windows_server/2016/english/full/recommended",
      "Type": "String",
      "Value": "{"schema_version":1,"image_name":"Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07","image_id":"ami-00c56e74f090d6f65","os":"Windows_Server-2016-English-Full","ecs_runtime_version":"Docker version 18.03.1-ee-7","ecs_agent_version":"1.26.0"}",
      "Version": 12,
      "LastModifiedDate": 1552024433.093,
    },
    "InvalidParameters": []
  ]
}
```

For the Amazon ECS-optimized Windows 2019 AMI: using AWS PowerShell

```

API Version 2014-11-13
95
```
### Retrieving Amazon ECS-Optimized AMI Metadata

#### Get-SSMParameter

```bash
Get-SSMParameter -Name /aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended/image_id -region us-east-1
```

#### Output:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>/aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended/image_id</td>
<td>String ami-04548e9 3</td>
</tr>
</tbody>
</table>

#### Example Retrieving the metadata of a specific Amazon ECS-optimized Amazon Linux 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. For more information about the available versions, see Amazon ECS-optimized AMI Versions (p. 82).

```bash
```

#### Example Retrieving the Amazon ECS-optimized Amazon Linux AMI metadata using the SSM GetParametersByPath API

Retrieve the Amazon ECS-optimized Amazon Linux AMI metadata with the SSM GetParametersByPath API using the AWS CLI with the following command.

```bash
aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux/ --region us-east-1
```

#### Example Retrieving the image ID of the latest recommended Amazon ECS-optimized Amazon Linux AMI

You can retrieve the image ID of the latest recommended Amazon ECS-optimized Amazon Linux AMI ID by using the sub-parameter `image_id`.

```bash
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux/recommended/image_id --region us-east-1
```

#### Output:

```json
{
  "Parameters": [
    {
      "Name": "/aws/service/ecs/optimized-ami/amazon-linux/recommended/image_id",
      "Type": "String",
      "Value": "ami-07eb698ce660402d2",
      "Version": 10,
      "LastModifiedDate": 1539892113.519,
      "ARN": "arn:aws:ssm:us-east-1::parameter/aws/service/ecs/optimized-ami/amazon-linux/recommended/image_id"
    }
  ],
```

API Version 2014-11-13
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/
recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

Output:

```
"ami-f9ac2f86"
```

**Example Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template**

You can retrieve the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template by referencing the SSM parameter store name; for example:

Amazon Linux 2:

```
Parameters:
ECSAMI:
  Description: AMI ID
  Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
  Default: /aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id
```

Windows:

```
Parameters:
ECSAMI:
  Description: AMI ID
  Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
  Default: /aws/service/ecs/optimized-ami/windows_server/2019/english/full/recommended/image_id
```

---

**Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications**

The Amazon ECS-optimized Amazon Linux AMI receives regular updates for agent changes, Docker version updates, and Linux kernel security updates. You can subscribe to the AMI update Amazon SNS topic to receive notifications when a new Amazon ECS-optimized Amazon Linux AMI is available. Notifications are available in all formats that Amazon SNS supports.

**Note**

Your user account must have `sns::subscribe` IAM permissions to subscribe to an SNS topic.

You can subscribe an Amazon SQS queue to this notification topic, but you must use a topic ARN that is in the same region. For more information, see Tutorial: Subscribing an Amazon SQS Queue to an Amazon SNS Topic in the Amazon Simple Queue Service Developer Guide.

You can also use an AWS Lambda function to trigger events when notifications are received. For more information, see Invoking Lambda functions using Amazon SNS notifications in the Amazon Simple Notification Service Developer Guide.
The Amazon SNS topic ARNs for each region are shown below.

<table>
<thead>
<tr>
<th>AWS Region</th>
<th>Amazon SNS Topic ARN</th>
</tr>
</thead>
</table>
Amazon Elastic Container Service Developer Guide
Amazon SNS Message Format

AWS Region | Amazon SNS Topic ARN
---|---

To subscribe to AMI update notification email in the AWS Management Console
1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v2/home.
2. In the region list, choose the same Region as the topic ARN to which to subscribe. This example uses the us-west-2 Region.
3. In the left navigation pane, choose Subscriptions, Create subscription.
5. For Protocol, choose Email. For Endpoint, type an email address that you can use to receive the notification.
6. Choose Create subscription.
7. In your email application, open the message from AWS Notifications and open the link to confirm your subscription.

Your web browser displays a confirmation response from Amazon SNS.

To subscribe to AMI update notification email with the AWS CLI
1. Run the following command with the AWS CLI:

   ```
   ```

   2. In your email application, open the message from AWS Notifications and open the link to confirm your subscription.

   Your web browser displays a confirmation response from Amazon SNS.

Amazon SNS Message Format
An example AMI update notification message is shown below:

```json
{
    "Type": "Notification",
    "MessageId": "e2653a930-337d-5561-8636-1a2be5ba802e",
```
The parsed Message value (with escaped quotes removed) is shown below:

```json
{
  "ECSAgent": {
    "ReleaseVersion": "1.17.2"
  },
  "ECSAmis": [
    {
      "ReleaseVersion": "2017.09.j",
      "AgentVersion": "1.17.2",
      "ReleaseNotes": "This AMI includes the latest ECS agent 1.17.2",
      "OsType": "linux",
      "OperatingSystemName": "Amazon Linux",
      "Regions": {
        "ap-northeast-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-bb5f13dd"
        },
        "ap-northeast-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-0356e07e"
        },
        "ap-south-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-da2c66b6"
        },
        "ap-southeast-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-9e91cff1"
        },
        "ap-southeast-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-64c4871d"
        },
        "ap-south-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-88ad8844"
        },
        "ap-northeast-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-3b19b455"
        },
        "ap-southeast-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-9e91cff1"
        },
        "ap-southeast-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-64c4871d"
        },
        "ca-central-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-1f46f0b7"
        },
        "eu-central-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-3b19b455"
        },
        "eu-west-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-64c4871d"
        },
        "eu-west-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-25f51242"
        },
        "eu-west-3": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-0356e07e"
        },
        "sa-east-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-da2c66b6"
        },
        "us-east-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-bb5f13dd"
        },
        "us-east-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-3b19b455"
        },
        "us-gov-west-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-64c4871d"
        },
        "us-west-1": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-9e91cff1"
        },
        "us-west-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-0356e07e"
        }
      }
    }
  ],
  "Timestamp": "2018-03-09T00:25:43.483Z",
  "SignatureVersion": "1",
  "Signature": "XWox8GDGLRlCgDOXlo/ fG9Lu/88P8S0FL6M6oQOMyfzskucuhoblsdea3BjqqChCwR7dNhMPQnLpN7y9iBrWUqAGJrukAIiathvAS+4AQD/ V/QjhshEnljGaiw +ozAu006X6GgopOzFgncFtPMr0jCMrMonjz7Hrvp/8KruM2ZjpyQQm5d4W67x8BPYU4IoLo5yVh8Fm5qaGqGQV/ YLnSY6u0U0BP1MttQaWdokC0vP3/Pv/yhGQlFxs19LqTadq1RIHtHIBWVL87PSI+BVk1in6AL7PnksydQ7FqAgHfXsIt +6PG59V9kCqaeE7HGNR1AbpyVka7JSNR0/6issyr1l1jg==",
}
```
Launching an Amazon ECS Container Instance

You can launch an Amazon ECS container instance using the AWS Management Console, as described in this topic. Before you begin, be sure that you’ve completed the steps in Setting Up with Amazon ECS (p. 7). After you’ve launched your instance, you can use it to run tasks.
To launch a container instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select the region to use.
3. From the console dashboard, choose Launch Instance.
4. On the Choose an Amazon Machine Image (AMI) page, complete the following steps:
   a. Choose Community AMIs.
   b. Choose an AMI for your container instance. You can choose the Amazon ECS-optimized Amazon Linux 2 AMI, or another operating system, such as CoreOS or Ubuntu. If you do not choose an Amazon ECS-optimized AMI, you must follow the procedures in Installing the Amazon ECS Container Agent (p. 133).

   **Note**
   For more information about Amazon ECS-specific CoreOS installation instructions, see Running CoreOS Container Linux with AWS EC2 Container Service.

To use the Amazon ECS-optimized Amazon Linux 2 AMI, type `amzn2-ami-ecs-hvm-2.0` in the Search community AMIs field and press the Enter key. Choose Select next to the `amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs` AMI.

The following table lists the current Amazon ECS-optimized Amazon Linux 2 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>ami-012ca23958772cf72</td>
</tr>
<tr>
<td>us-east-1</td>
<td>ami-00cf4737e23866a3</td>
</tr>
<tr>
<td>us-west-1</td>
<td>ami-06d87f0156b1d4407</td>
</tr>
<tr>
<td>us-west-2</td>
<td>ami-0a9f5be2a016dccad</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>ami-067f4f7124e746edd</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>ami-0e52aad6ac7733a6a</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>ami-08834c857e502d6d</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>ami-04322e867758d97a8</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>ami-0047fd6b16f1f6781</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>ami-09475847322e5566f</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>ami-0498c646ec4d2ba83</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>ami-0c5ab45f676aaf4f</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>ami-0c5ab45f676aaf4f</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>ami-0204aa6a92a54561e</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>ami-07273195833e4f20c</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>ami-096a38c97b80cd8ec</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>ami-0ddc4daca44e6e99</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>ami-00d851648873aaabc</td>
</tr>
</tbody>
</table>
### Region AMI ID

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-gov-east-1</td>
<td>ami-06473be43b0f77600</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>ami-607c0001</td>
</tr>
</tbody>
</table>

**Note**

The current Amazon ECS-optimized Amazon Linux 2 AMI ID can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/recommended
```

5. On the **Choose an Instance Type** page, you can select the hardware configuration of your instance. The `t2.micro` instance type is selected by default. The instance type that you select determines the resources available for your tasks to run on.

Choose **Next: Configure Instance Details** when you are done.

6. On the **Configure Instance Details** page, complete the following steps:

   a. Set the **Number of instances** field depending on how many container instances you want to add to your cluster.

   b. (Optional) To use Spot Instances, for **Purchasing option**, select the check box next 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.

   c. For **Network**, choose the VPC into which to launch your container instance.

   d. For **Subnet**, choose a subnet to use, or keep the default option to choose the default subnet in any Availability Zone.

   e. Set the **Auto-assign Public IP** field depending on whether you want your instance to be accessible from the public internet. 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 external network access to communicate with the Amazon ECS service endpoint, so if 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 [HTTP Proxy Configuration](#) in this guide. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters](#).

   f. Select the `ecsInstanceRole` **IAM role** value that you created for your container instances in [Setting Up with Amazon ECS](#).

      **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](#).

   g. (Optional) 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.

```bash
#!/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 Container Instance Configuration in Amazon S3 (p. 163).

```bash
#!/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.

```bash
#!/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:

```bash
#!/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 Container Instances with Amazon EC2 User Data (p. 105).

h. Choose Next: Add Storage.

7. On the Add Storage page, 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.

When done configuring your volumes, choose **Next: Add Tags**.

8. On the **Add Tags** page, specify tags by providing key and value combinations for the container instance. Choose **Add another tag** to add more than one tag to your container instance. For more information resource tags, see Resources and Tags (p. 355).

Choose **Next: Configure Security Group** when you are done.

9. On the **Configure Security Group** page, 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. Select or create a security group as follows, and then choose **Review and Launch**.

10. On the **Review Instance Launch** page, under **Security Groups**, you see that the wizard created and selected a security group for you. Instead, select the security group that you created in Setting Up with Amazon ECS (p. 7) using the following steps:

   a. Choose **Edit security groups**.
   b. On the **Configure Security Group** page, select the **Select an existing security group** option.
   c. Select the security group you created for your container instance from the list of existing security groups, and choose **Review and Launch**.

11. On the **Review Instance Launch** page, choose **Launch**.

12. In the **Select an existing key pair or create a new key pair** dialog box, choose **Choose an existing key pair**, then select the key pair that you created when getting set up.

When you are ready, select the acknowledgment field, and then choose **Launch Instances**.

13. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.

14. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is **pending**. After the instance starts, its state changes to **running**, and it receives a public DNS name. If the **Public DNS** column is hidden, choose **Show/Hide, Public DNS**.

---

**Bootstrapping Container Instances with Amazon EC2 User Data**

When you launch an Amazon ECS container instance, you have the option of passing user data to the 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 into the Amazon EC2 launch wizard in Step 6.g (p. 103) of Launching an Amazon ECS Container Instance (p. 101).
## 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** (p. 152).

To set only a single agent configuration variable, such as the cluster name, use `echo` to copy the variable to the configuration file:

```bash
#!/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 `here doc` format. This format writes everything between the lines beginning with `cat` and `EOF` to the configuration file.

```bash
#!/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
EOF
```

## Docker Daemon

You can specify Docker daemon configuration information with Amazon EC2 user data, but this configuration data must be written before the Docker daemon starts. The `cloud-boothook` user data format executes earlier in the boot process than a user data shell script. For more information about configuration options, see the Docker daemon documentation.

By default, `cloud-boothook` user data is run at every instance boot, so you must create a mechanism to prevent the boothook from running multiple times. The `cloud-init-per` utility is provided to control boothook frequency in this manner. For more information, see **cloud-init-per Utility** (p. 107).

In the example below, the `--foo bar` option is appended to any existing options in the Docker daemon configuration file, `/etc/sysconfig/docker`.

```
#cloud-boothook
cloud-init-per once docker_options echo 'OPTIONS="$\{OPTIONS\} --foo bar"' >> /etc/sysconfig/docker
```

To write multiple lines to a file, use the following `here doc` format to accomplish the same goal:

```
#cloud-boothook
cloud-init-per instance docker_options cat <<'EOF' >> /etc/sysconfig/docker
OPTIONS="$\{OPTIONS\} --foo bar"
EOF
```
cloud-init-per Utility

The **cloud-init-per** utility is provided by the `cloud-init` package to help you create boothook commands for instances that run at a specified frequency.

The **cloud-init-per** utility syntax is as follows:

```
cloud-init-per frequency name cmd [ arg1 [ arg2 [ ... ] ]
```

- **frequency**
  - How often the boothook should run.
  - Specify `once` to never run again, even with a new instance ID.
  - Specify `instance` to run on the first boot for each new instance launch. For example, if you create an AMI from the instance after the boothook has run, it still runs again on subsequent instances launched from that AMI.
  - Specify `always` to run at every boot.

- **name**
  - The name to include in the semaphore file path that is written when the boothook runs. The semaphore file is written to `/var/lib/cloud/instances/instance_id/sem/bootper.name.instance`.

- **cmd**
  - The command and arguments that the boothook should execute.

In the example below, the command `echo 'OPTIONS="${OPTIONS} --foo bar"' >> /etc/sysconfig/docker` is executed only once. A semaphore file is written that contains its name.

```
#cloud-boothook
cloud-init-per once docker_options echo 'OPTIONS="$(OPTIONS) --foo bar"' >> /etc/sysconfig/docker
```

The semaphore file records the exit code of the command and a UNIX timestamp for when it was executed.

```
[ec2-user ~]$ cat /var/lib/cloud/instances/i-0c7f87d7611b2165e/sem/bootper.docker_options.instance
```

Output:

```
0 1488410363
```

**Specifying Multiple User Data Blocks Using a MIME Multi Part Archive**

You can combine multiple user data blocks together into a single user data block called a MIME multi-part file. For example, you might want to combine a cloud boothook that configures the Docker daemon with a user data shell script that writes configuration information for the Amazon ECS container agent.
A MIME multi-part file consists of the following components:

- The content type and part boundary declaration: `Content-Type: multipart/mixed; boundary="==BOUNDARY=="`
- The MIME version declaration: `MIME-Version: 1.0`
- One or more user data blocks, which contain the following components:
  - The opening boundary, which signals the beginning of a user data block: `--==BOUNDARY==`
  - The content type declaration for the block: `Content-Type: text/cloud-boothook; charset="us-ascii"`. For more information about content types, see the Cloud-Init documentation.
  - The content of the user data, for example, a list of shell commands or cloud-init directives
  - The closing boundary, which signals the end of the MIME multi-part file: `--==BOUNDARY==--`

Example MIME multi-part file

This example MIME multi-part file configures the Docker base device size to 20 GiB and configures the Amazon ECS container agent to register the instance into the cluster named `my-ecs-cluster`.

```
Content-Type: multipart/mixed; boundary="==BOUNDARY=="
MIME-Version: 1.0

--==BOUNDARY==
Content-Type: text/cloud-boothook; charset="us-ascii"

# Set Docker daemon options
cloud-init-per once docker_options echo 'OPTIONS="${OPTIONS} --foo bar"' >> /etc/sysconfig/docker

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
# Set any ECS agent configuration options
echo "ECS_CLUSTER="my-ecs-cluster"" >> /etc/ecs/ecs.config

--==BOUNDARY==--
```

Example Container Instance User Data Configuration Scripts

The following example user data scripts configure an Amazon ECS container instance at launch.

**Amazon ECS-Optimized Amazon Linux AMI Container Instance with Amazon EFS File System**

This example user data script configures an instance launched from the Amazon ECS-optimized Amazon Linux AMI to use an existing Amazon EFS file system. For more information, see Tutorial: Using Amazon EFS File Systems with Amazon ECS (p. 581)

This script does the following:

- Install the `nfs-utils` package, which installs an NFS client.
- Create a mount directory for the NFS file system at `/efs`. 
- Create a mount entry in the `/etc/fstab` file for the file system and then mount the file system.
- Write the cluster name, `default`, to the Amazon ECS agent configuration file.

You can use this script for your own container instances, provided that they are launched from an Amazon ECS-optimized Amazon Linux AMI. Be sure to replace the `ECS_CLUSTER=default` line in the configuration file to specify your own cluster name, if you are not using the default cluster. For more information about launching container instances, see Launching an Amazon ECS Container Instance (p. 101).

| Content-Type: multipart/mixed; boundary="==BOUNDARY==" |
| MIME-Version: 1.0 |
| --==BOUNDARY== |
| Content-Type: text/cloud-boothook; charset="us-ascii" |

```
# Install nfs-utils
cloud-init-per once yum_update yum update -y
cloud-init-per once install_nfs_utils yum install -y nfs-utils

# Create /efs folder
cloud-init-per once mkdir_efs mkdir /efs

# Mount /efs
cloud-init-per once mount_efs echo -e '
fs-abcd1234.efs.us-east-1.amazonaws.com:/efs nfs4
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 0 0' >> /etc/fstab
mount -a
```

| Content-Type: text/x-shellscript; charset="us-ascii" |
| --==BOUNDARY== |

```
#!/bin/bash
# Set any ECS agent configuration options
echo "ECS_CLUSTER=default" >> /etc/ecs/ecs.config
```

Ubuntu Container Instance with systemd

This example user data script configures an Ubuntu 16.04 instance to:

- Install Docker.
- Create the required `iptables` rules for IAM roles for tasks.
- Create the required directories for the Amazon ECS container agent.
- Write the Amazon ECS container agent configuration file.
- Write the `systemd` unit file to monitor the agent.
- Enable and start the `systemd` unit.

You can use this script for your own container instances, provided that they are launched from an Ubuntu 16.04 AMI. Be sure to replace the `ECS_CLUSTER=default` line in the configuration file to specify your own cluster name, if you are not using the default cluster. For more information about launching container instances, see Launching an Amazon ECS Container Instance (p. 101).

```
#!/bin/bash
# Install Docker
apt-get update -y && apt-get install -y docker.io
```

API Version 2014-11-13

109
# Set iptables rules

```bash
echo 'net.ipv4.conf.all.route_localnet = 1' >> /etc/sysctl.conf
sysctl -p /etc/sysctl.conf
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
```

# Write iptables rules to persist after reboot

```bash
iptables-save > /etc/iptables/rules.v4
```

# Create directories for ECS agent

```bash
mkdir -p /var/log/ecs /var/lib/ecs/data /etc/ecs
```

# Write ECS config file

```bash
cat << EOF > /etc/ecs/ecs.config
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECSAVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
EOF
```

# Write systemd unit file

```bash
cat << EOF > /etc/systemd/system/docker-container@ecs-agent.service
[Unit]
Description=Docker Container %i
Requires=docker.service
After=docker.service

[Service]
Restart=always
ExecStartPre=-/usr/bin/docker rm -f %i
ExecStart=/usr/bin/docker run --name %i --restart=on-failure:10 --volume=/var/run:/var/run --volume=/var/lib/ecs:/log --volume=/var/lib/ecs/data:/data --volume=/etc/ecs:/etc/ecs --net=host --env-file=/etc/ecs/ecs.config \amazon/amazon-ecs-agent:latest
ExecStop=/usr/bin/docker stop %i

[Install]
WantedBy=default.target
EOF
```

```bash
systemctl enable docker-container@ecs-agent.service
systemctl start docker-container@ecs-agent.service
```

## CentOS Container Instance with systemd and SELinux

This example user data script configures a CentOS 7 instance with SELinux enabled to:

- Install Docker.
- Create the required `iptables` rules for IAM roles for tasks.
- Create the required directories for the Amazon ECS container agent.
- Write the Amazon ECS container agent configuration file.
- Write the `systemd` unit file to monitor the agent.
• Enable and start the **systemd** unit.

**Note**
The **docker run** command in the **systemd** unit file below contains the required modifications for SELinux, including the **--privileged** flag, and the :Z suffixes to the volume mounts.

You can use this script for your own container instances (provided that they are launched from an CentOS 7 AMI). Be sure to replace the **ECS_CLUSTER=default** line in the configuration file to specify your own cluster name (if you are not using the default cluster). For more information about launching container instances, see Launching an Amazon ECS Container Instance (p. 101).

```bash
#!/bin/bash
# Install Docker
yum install -y docker

# Set iptables rules
echo 'net.ipv4.conf.all.route_localnet = 1' >> /etc/sysctl.conf
sysctl -p /etc/sysctl.conf
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

# Write iptables rules to persist after reboot
iptables-save > /etc/sysconfig/iptables

# Create directories for ECS agent
mkdir -p /var/log/ecs /var/lib/ecs/data /etc/ecs

# Write ECS config file
cat << EOF > /etc/ecs/ecs.config
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
EOF

# Write systemd unit file
if [[ -f /etc/systemd/system/docker-container@ecs-agent.service ]]; then
    echo '#!/bin/bash
    # Install Docker
    yum install -y docker
    # Set iptables rules
echo 'net.ipv4.conf.all.route_localnet = 1' >> /etc/sysctl.conf
    sysctl -p /etc/sysctl.conf
    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
    # Write iptables rules to persist after reboot
    iptables-save > /etc/sysconfig/iptables
    # Create directories for ECS agent
    mkdir -p /var/log/ecs /var/lib/ecs/data /etc/ecs
    # Write ECS config file
cat << EOF > /etc/ecs/ecs.config
    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
    EOF
    # Write systemd unit file
cat << EOF > /etc/systemd/system/docker-container@ecs-agent.service
    [Unit]
    Description=Docker Container %I
    Requires=docker.service
    After=docker.service
    [Service]
    Restart=always
    ExecStartPre=-/usr/bin/docker rm -f %i
    ExecStart=/usr/bin/docker run --name %i --privileged --restart=on-failure:10 --volume=/var/run:/var/run --volume=/var/log/ecs:/log:Z --volume=/var/lib/ecs/data:/data:Z --volume=/etc/ecs:/etc/ecs --net=host --env-file=/etc/ecs/ecs.config amazon/amazon-ecs-agent:latest
    ExecStop=/usr/bin/docker stop %i
    [Install]
```
Default Windows User Data

This example user data script shows the default user data that your Windows container instances receive if you use the cluster creation wizard (p. 55). The below script does the following:

- Sets the cluster name to `windows`.
- Enables IAM roles for tasks.
- Sets `json-file` and `awslogs` as the available logging drivers.

You can use this script for your own container instances (provided that they are launched from the Amazon ECS-optimized Windows AMI), but be sure to replace the `-Cluster windows` line to specify your own cluster name (if you are not using a cluster called `windows`).

```powershell
<powershell>
Initialize-ECSAgent -Cluster windows -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 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 AMI. For more information, see Creating a Cluster (p. 55).

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", "${env:ProgramData}\Amazon\ECS\logs\ecs-agent.log", "Machine")
[Environment]::SetEnvironmentVariable("ECS_DATADIR", "${env:ProgramData}\Amazon\ECS\data", "Machine")
# Download the agent
#agentVersion = "latest"
#zipFile = "${env:TEMP}\ecs-agent.zip"
Invoke-RestMethod -OutFile $zipFile -Uri "agentZipUri" -Put $executables in the executable directory.
Expand-Archive -Path $zipFile -DestinationPath $ecsExeDir -Force
```

API Version 2014-11-13

112
Connect to Your Container Instance

To perform basic administrative tasks on your instance, such as updating or installing software or accessing diagnostic logs, connect to the instance using SSH. To connect to your instance using SSH, your container instances must meet the following prerequisites:

- Your container instances need external network access to connect using SSH. If your container instances are running in a private VPC, they need an SSH bastion instance to provide this access. For more information, see the Securely connect to Linux instances running in a private Amazon VPC blog post.
- Your container instances must have been launched with a valid Amazon EC2 key pair. Amazon ECS container instances have no password, and you use a key pair to log in using SSH. If you did not specify a key pair when you launched your instance, there is no way to connect to the instance.
- SSH uses port 22 for communication. Port 22 must be open in your container instance security group for you to connect to your instance using SSH.

Note
The Amazon ECS console first-run experience creates a security group for your container instances without inbound access on port 22. If your container instances were launched from the console first-run experience, add inbound access to port 22 on the security group used for those instances. For more information, see Authorizing Network Access to Your Instances in the Amazon EC2 User Guide for Linux Instances.

To connect to your container instance

1. Find the public IP or DNS address for your container instance.
   a. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
   b. Select the cluster that hosts your container instance.
   c. On the Cluster page, choose ECS Instances.
   d. On the Container Instance column, select the container instance to connect to.
   e. On the Container Instance page, record the Public IP or Public DNS for your instance.
2. Find the default username for your container instance AMI. The user name for instances launched with an Amazon ECS-optimized AMI is ec2-user. For Ubuntu AMIs, the default user name is ubuntu. For CoreOS, the default user name is core.
3. 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 Connecting to Your Linux Instance from Windows Using PuTTY in the Amazon EC2 User Guide for Linux Instances.

**Important**
For more information about any issues while connecting to your instance, see Troubleshooting Connecting to Your Instance in the Amazon EC2 User Guide for Linux Instances.

### Using CloudWatch Logs with Container Instances

You can configure your container instances to send log information to CloudWatch Logs. This enables you to view different logs from your container instances in one convenient location. This topic helps you get started using CloudWatch Logs on your container instances that were launched with the Amazon ECS-optimized Amazon Linux AMI.

For information about sending container logs from your tasks to CloudWatch Logs, see Using the awsllogs Log Driver (p. 245). For more information about CloudWatch Logs, see Monitoring Log Files in the Amazon CloudWatch User Guide.

**Topics**
- CloudWatch Logs IAM Policy (p. 114)
- Installing the CloudWatch Logs Agent (p. 115)
- Configuring and Starting the CloudWatch Logs Agent (p. 115)
- Viewing CloudWatch Logs (p. 118)
- Configuring CloudWatch Logs at Launch with User Data (p. 119)

### CloudWatch Logs IAM Policy

Before your container instances can send log data to CloudWatch Logs, you must create an IAM policy to allow your container instances to use the CloudWatch Logs APIs, and then you must attach that policy to ecsInstanceRole.

**To create the ECS-CloudWatchLogs IAM policy**

2. In the navigation pane, choose Policies.
3. Choose Create policy, JSON.
4. Enter the following policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [
```
5. Choose **Review policy**.

6. On the **Review policy** page, enter `ECS-CloudWatchLogs` for the **Name** and choose **Create policy**.

**To attach the ECS-CloudWatchLogs policy to ecsInstanceRole**

2. In the navigation pane, choose **Roles**.
3. Choose `ecsInstanceRole`. If the role does not exist, follow the procedures in Amazon ECS Container Instance IAM Role (p. 415) to create the role.
4. Choose **Permissions, Attach policy**.
5. To narrow the available policies to attach, for **Filter**, type `ECS-CloudWatchLogs`.
6. Check the box to the left of the `ECS-CloudWatchLogs` policy and choose **Attach policy**.

### Installing the CloudWatch Logs Agent

After you have added the ECS-CloudWatchLogs policy to your `ecsInstanceRole`, you can install the CloudWatch Logs agent on your container instances.

**Note**

This procedure was written for the Amazon ECS-optimized Amazon Linux AMI, and may not work on other operating systems. For information about installing the agent on other operating systems, see Getting Started with CloudWatch Logs in the Amazon CloudWatch User Guide.

**To install the CloudWatch Logs agent**

- Run the following command to install the CloudWatch Logs agent.

  ```sh
  [ec2-user ~]# sudo yum install -y awslogs
  ```

After you have installed the agent, proceed to the next section to configure the agent.

### Configuring and Starting the CloudWatch Logs Agent

The CloudWatch Logs agent configuration file (`/etc/awslogs/awslogs.conf`) describes the log files to send to CloudWatch Logs. The agent configuration file's `[general]` section defines common configurations that apply to all log streams, and you can add individual log stream sections for each file on your container instances that you want to monitor. For more information, see CloudWatch Logs Agent Reference in the Amazon CloudWatch User Guide.

The example configuration file below is configured for the Amazon ECS-optimized Amazon Linux AMI, and it provides log streams for several common log files:
/var/log/dmesg
   The message buffer of the Linux kernel.
/var/log/messages
   Global system messages.
/var/log/docker
   Docker daemon log messages.

   Note
   This log file path is only relevant to instances based on the Amazon Linux AMI. For Amazon Linux 2, Docker logs are stored in journald, which is not currently supported by CloudWatch Logs.
/var/log/ecs/ecs-init.log
   Log messages from the ecs-init service.
/var/log/ecs/ecs-agent.log
   Log messages from the Amazon ECS container agent.
/var/log/ecs/audit.log
   Log messages from the IAM roles for the task credential provider.

You can use the example file below for your Amazon ECS container instances, but you must substitute the {cluster} and {container_instance_id} entries with the cluster name and container instance ID for each container instance so that the log streams are grouped by cluster name and separate for each individual container instance. The procedure that follows the example configuration file has steps to replace the cluster name and container instance ID placeholders.

[general]
state_file = /var/lib/awslogs/agent-state

[/var/log/dmesg]
file = /var/log/dmesg
log_group_name = /var/log/dmesg
log_stream_name = {cluster}/{container_instance_id}

[/var/log/messages]
file = /var/log/messages
log_group_name = /var/log/messages
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %b %d %H:%M:%S

[/var/log/docker]
file = /var/log/docker
log_group_name = /var/log/docker
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%d%H:%M:%S

[/var/log/ecs/ecs-init.log]
file = /var/log/ecs/ecs-init.log
log_group_name = /var/log/ecs/ecs-init.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

[/var/log/ecs/ecs-agent.log]
file = /var/log/ecs/ecs-agent.log
log_group_name = /var/log/ecs/ecs-agent.log
log_stream_name = {cluster}/{container_instance_id}
To configure the CloudWatch Logs agent

1. Back up the existing CloudWatch Logs agent configuration file.

```
[ec2-user ~]# sudo mv /etc/awslogs/awslogs.conf /etc/awslogs/awslogs.conf.bak
```

2. Create a blank configuration file.

```
[ec2-user ~]# sudo touch /etc/awslogs/awslogs.conf
```

3. Open the `/etc/awslogs/awslogs.conf` file with a text editor, and copy the example file above into it.

4. Install the `jq` JSON query utility.

```
[ec2-user ~]# sudo yum install -y jq
```

5. Query the Amazon ECS introspection API to find the cluster name and set it to an environment variable.

```
[ec2-user ~]# cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '.Cluster')
```

6. Replace the `{cluster}` placeholders in the file with the value of the environment variable you set in the previous step.

```
[ec2-user ~]# sudo sed -i -e "s/{cluster}/$cluster/g" /etc/awslogs/awslogs.conf
```

7. Query the Amazon ECS introspection API operation to find the container instance ID and set it to an environment variable.

```
[ec2-user ~]# container_instance_id=$(curl -s http://localhost:51678/v1/metadata | jq -r '.ContainerInstanceArn' | awk -F/ '{print $2}')
```

8. Replace the `{container_instance_id}` placeholders in the file with the value of the environment variable you set in the previous step.

```
[ec2-user ~]# sudo sed -i -e "s/{container_instance_id}/$container_instance_id/g" /etc/awslogs/awslogs.conf
```

To configure the CloudWatch Logs agent Region

By default, the CloudWatch Logs agent sends data to the `us-east-1` region. To send your data to a different region, such as the Region in which your cluster is located, you can set the Region in the `/etc/awslogs/awscli.conf` file.

1. Open the `/etc/awslogs/awscli.conf` file with a text editor.
2. In the `[default]` section, replace `us-east-1` with the Region from which to view log data.
3. Save the file and exit your text editor.
To start the CloudWatch Logs agent

1. Start the CloudWatch Logs agent with the following command.
   - Amazon Linux
     
     [ec2-user ~]$ sudo service awslogs start
   - Amazon Linux 2
     
     [ec2-user ~]$ sudo systemctl start awslogsd

2. Ensure that the CloudWatch Logs agent starts at every system boot with the following command.
   - Amazon Linux
     
     [ec2-user ~]$ sudo chkconfig awslogs on
   - Amazon Linux 2
     
     [ec2-user ~]$ sudo systemctl enable awslogsd.service

Viewing CloudWatch Logs

After you have given your container instance role the proper permissions to send logs to CloudWatch Logs, and you have configured and started the agent, your container instance should be sending its log data to CloudWatch Logs. You can view and search these logs in the AWS Management Console.

Note
New instance launches may take a few minutes to send data to CloudWatch Logs.

To view your CloudWatch Logs data

2. In the left navigation pane, choose Logs.

   You should see the log groups you configured in Configuring and Starting the CloudWatch Logs Agent (p. 115).

   Log Groups

   ![Create Metric Filter](image)

   ![Actions](image)

   Filter: Log Group Name Prefix

   Log Groups
   - /var/log/dmesg
   - /var/log/docker
   - /var/log/ecs/ecs-agent.log
   - /var/log/ecs/ecs-init.log
   - /var/log/messages

3. Choose a log group to view.
4. Choose a log stream to view. The streams are identified by the cluster name and container instance ID that sent the logs.
When you launch an Amazon ECS container instance in Amazon EC2, you have the option of passing user data to the instance that can be used to perform common automated configuration tasks and even run scripts after the instance starts. You can pass several types of user data to instances, including shell scripts, `cloud-init` directives, and system services. You can also pass this data into the launch wizard as plaintext, as a file (this is useful for launching instances via the command line tools), or as base64-encoded text (for API calls).

The example user data blocks shown below (note that there are separate user data blocks for Amazon Linux 2 and Amazon Linux AMI variants) perform the following tasks:

- Installs the `awslogs` package, which contains the CloudWatch Logs agent
- Installs the `jq` JSON query utility
- Writes the configuration file for the CloudWatch Logs agent and configures the Region to send data to (the Region in which the container instance is located)
- Gets the cluster name and container instance ID after the Amazon ECS container agent starts and then writes those values to the CloudWatch Logs agent configuration file log streams
- Starts the CloudWatch Logs agent
- Configures the CloudWatch Logs agent to start at every system boot

**Example Amazon ECS-optimized Amazon Linux 2 AMI user data**

```bash
Content-Type: multipart/mixed; boundary="==BOUNDARY=="

MIME-Version: 1.0

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/usr/bin/env bash
# Install awslogs and the jq JSON parser
yum install -y awslogs jq

# Inject the CloudWatch Logs configuration file contents
cat > /etc/awslogs/awslogs.conf <<- EOF
[general]
state_file = /var/lib/awslogs/agent-state

[/var/log/dmesg]
file = /var/log/dmesg
log_group_name = /var/log/dmesg
log_stream_name = {cluster}/{container_instance_id}

[/var/log/messages]
EOF
```

API Version 2014-11-13

119
file = /var/log/messages
log_group_name = /var/log/messages
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %b %d %H:%M:%S

file = /var/log/ecs/ecs-init.log
log_group_name = /var/log/ecs/ecs-init.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

file = /var/log/ecs/ecs-agent.log
log_group_name = /var/log/ecs/ecs-agent.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

file = /var/log/ecs/audit.log
log_group_name = /var/log/ecs/audit.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%SZ

---==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/usr/bin/env bash
# Write the awslogs bootstrap script to /usr/local/bin/bootstrap-awslogs.sh
cat > /usr/local/bin/bootstrap-awslogs.sh <<- 'EOF'
#!/usr/bin/env bash
exec 2>>/var/log/ecs/cloudwatch-logs-start.log
set -x
until curl -s http://localhost:51678/v1/metadata
do
  sleep 1
done

# Set the region to send CloudWatch Logs data to (the region where the container instance
# is located)
region=${(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r .region)
set -e "s/region = .*/region = "$region" /etc/awslogs/awscli.conf"

# Grab the cluster and container instance ARN from instance metadata
cluster=${(curl -s http://localhost:51678/v1/metadata | jq -r .Cluster)
container_instance_id=${(curl -s http://localhost:51678/v1/metadata | jq -r .ContainerInstanceArn) | awk -F/ '{print $2}')

# Replace the cluster name and container instance ID placeholders with the actual values
EOF

---==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/usr/bin/env bash
# Write the bootstrap-awslogs systemd unit file to /etc/systemd/system/bootstrap-awslogs.service
cat > /etc/systemd/system/bootstrap-awslogs.service <<- EOF
[Unit]
Description=Bootstrap awslogs agent
Requires=ecs.service
After=ecs.service
EOF
Before=awslogsd.service

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/local/bin/bootstrap-awslogs.sh

[Install]
WantedBy=awslogsd.service

EOF

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/sh
chmod +x /usr/local/bin/bootstrap-awslogs.sh
systemctl daemon-reload
systemctl enable bootstrap-awslogs.service
systemctl enable awslogsd.service
systemctl start awslogsd.service --no-block

--==BOUNDARY==--

Example Amazon ECS-optimized Amazon Linux AMI user data

Content-Type: multipart/mixed; boundary="==BOUNDARY=="
MIME-Version: 1.0

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/usr/bin/env bash

# Install awslogs and the jq JSON parser
yum install -y awslogs jq

# Inject the CloudWatch Logs configuration file contents

# Install awslogs and the jq JSON parser

# Inject the CloudWatch Logs configuration file contents

cat > /etc/awslogs/awslogs.conf <<- EOF
[general]
state_file = /var/lib/awslogs/agent-state

[/var/log/dmesg]
file = /var/log/dmesg
log_group_name = /var/log/dmesg
log_stream_name = {cluster}/{container_instance_id}

datetime_format = %b %d %H:%M:%S

[/var/log/messages]
file = /var/log/messages
log_group_name = /var/log/messages
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %b %d %H:%M:%S

[/var/log/docker]
file = /var/log/docker
log_group_name = /var/log/docker
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%S.%f

[/var/log/ecs/ecs-init.log]
file = /var/log/ecs/ecs-init.log
log_group_name = /var/log/ecs/ecs-init.log
log_stream_name = {cluster}/{container_instance_id}
datetime_format = %Y-%m-%dT%H:%M:%S

[/var/log/ecs/ecs-agent.log]
file = /var/log/ecs/ecs-agent.log.*
log_group_name = /var/log/ecs/ecs-agent.log

EOF

--==BOUNDARY==--
Container Instance Draining

There are times when you might need to remove a container instance from a cluster; for example, to perform system updates, update the Docker daemon, or scale down the cluster size. Container instance
draining enables you to remove a container instance from a cluster without impacting tasks in your cluster.

When you set a container instance 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.

Service tasks on the container instance that are in the RUNNING state are stopped and replaced according to the service's deployment configuration parameters, minimumHealthyPercent and maximumPercent.

- 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.

- The maximumPercent parameter represents an upper limit on the number of running tasks during task replacement, which enables 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.

For more information, see Service Definition Parameters (p. 292).

Any PENDING or RUNNING tasks that do not belong to a service are unaffected; you must wait for them to finish or stop them manually.

A container instance has completed draining when there are no more RUNNING tasks (although the state remains as DRAINING). You can verify this using the ListTasks operation with the containerInstance parameter.

When you change the status of a container instance from DRAINING to ACTIVE, the Amazon ECS scheduler can schedule tasks on the instance again.

## Draining Instances

You can use the UpdateContainerInstancesState API action or the update-container-instances-state command to change the status of a container instance to DRAINING.

The following procedure demonstrates how to set your instance to DRAINING using the AWS Management Console.

**To set your instance to DRAINING using the console**

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Clusters and select the cluster.
3. Choose ECS Instances and select the check box for the container instances.
4. Choose Actions, Drain instances.
5. After the instances are processed, choose Done.
Container Instance Memory Management

When the Amazon ECS container agent registers a container instance into 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.

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.

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 trigger a system failure. For more information, see Reserving System Memory (p. 124).

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.

```bash
$ 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.

```cmd
C:\> wmic ComputerSystem get TotalPhysicalMemory
```

Example output for an m4.large instance running the Amazon ECS-optimized Windows AMI.

```
TotalPhysicalMemory
8589524992
```

This instance has 8589524992 bytes of total memory, which translates to 8191 MiB available for tasks.

**Reserving System Memory**

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 trigger a system failure. 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.

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 (p. 152) and Bootstrapping Container Instances with Amazon EC2 User Data (p. 105).

### 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 Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster that hosts your container instances to view.
3. Choose **ECS Instances**, and select a container instance from the **Container Instance** column to view.
4. The **Resources** section shows the registered and available memory for the container instance.

#### Resources

<table>
<thead>
<tr>
<th>Resources</th>
<th>Registered</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>Memory</td>
<td>7953</td>
<td>7953</td>
</tr>
<tr>
<td>Ports</td>
<td>5 ports</td>
<td></td>
</tr>
</tbody>
</table>

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 Container Instances Remotely

You can use the Amazon EC2 Run Command feature to securely and remotely manage the configuration of your Amazon ECS container instances. Run Command provides a simple way of performing common administrative tasks without having to log 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.

This topic covers basic installation of Run Command on the Linux variants of the Amazon ECS-optimized AMI and a few simple use cases, but it is by no means exhaustive. For more information about Run Command, see Manage Amazon EC2 Instances Remotely in the Amazon EC2 User Guide for Linux Instances.

Topics
• Run Command IAM Policy (p. 126)
• Installing the SSM Agent on an Amazon ECS-Optimized AMI (p. 126)
• Using Run Command (p. 127)

Run Command IAM Policy

Before you can send commands to your container instances with Run Command, you must attach an IAM policy that allows access to the Amazon EC2 Systems Manager (SSM) APIs to the ecsInstanceRole.

The procedure below describes how to attach the AmazonEC2RoleforSSM managed policy to your container instance role so that instances launched with this role can use Run Command.

To attach the AmazonEC2RoleforSSM policy to your ecsInstanceRole
1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Choose ecsInstanceRole. If the role does not exist, follow the procedures in Amazon ECS Container instance IAM Role (p. 415) to create the role.
4. Choose Permissions.
5. In the Managed Policies section, choose Attach Policy.
6. To narrow the available policies to attach, for Filter, type AmazonEC2RoleforSSM.
7. Select the check box for the AmazonEC2RoleforSSM policy and choose Attach Policy.

Installing the SSM Agent on an Amazon ECS-Optimized AMI

After you have attached the AmazonEC2RoleforSSM policy to your ecsInstanceRole, you can install the SSM agent on your container instances. The SSM agent processes Run Command requests and configures the instances that are specified in the request. Use the following procedures to install the SSM agent on your Amazon ECS-optimized AMI container instances.

To manually install the SSM agent on existing Amazon ECS-optimized AMI container instances
1. Connect to your container instance. (p. 113)
2. Install the SSM agent RPM. The SSM agent is available in all Regions that Amazon ECS is available in. Each Region has its own region-specific download URL. The example command below works for all Regions that Amazon ECS supports. Avoid cross-region data transfer costs for the RPM download by substituting the Region of your container instance.

```
```

To install the SSM agent on new instance launches with Amazon EC2 user data

- Launch one or more container instances by following the procedure in Launching an Amazon ECS Container Instance (p. 101), but in Step 6.g (p. 103), copy and paste the user data script below into the User data field. You can also add the commands from this user data script to another existing script that you may have to perform other tasks, such as setting the cluster name for the instance to register into.

  **Note**
  The user data script below installs the jq JSON parser and uses that to determine the region of the container instance. Then it downloads and installs the SSM agent.

```
#!/bin/bash
# Install JQ JSON parser
yum install -y jq

# Get the current region from the instance metadata

# Install the SSM agent RPM
```

**Using Run Command**

After you have attached the AmazonEC2RoleforSSM policy to your ecsInstanceRole, and installed the SSM agent on your container instances, you can start using Run Command to send commands to your container instances. The following topic in the Amazon EC2 User Guide for Linux Instances explains how to run commands and shell scripts on your instances and view the resulting output:

- Running Shell Scripts with Run Command

For more information about Run Command, see Manage Amazon EC2 Instances Remotely in the Amazon EC2 User Guide for Linux Instances.

**Example: To update container instance software with Run Command**

A common use case for Run Command is to update the instance software on your entire fleet of container instances at one time.

1. Attach the AmazonEC2RoleforSSM policy to your ecsInstanceRole. (p. 126)
2. Install the SSM agent on your container instances. For more information, see Installing the SSM Agent on an Amazon ECS-Optimized AMI (p. 126).
3. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
4. In the left navigation, choose Commands, Run a command.
5. For **Command document**, choose **AWS-RunShellScript**.
6. In the **Target instances** section, choose **Select instances** and check the container instances to which to send the update command.
7. In the **Commands** section, enter the command or commands to send to your container instances. In this example, the command below updates the instance software:

```bash
# yum update -y
```

8. Choose **Run** to send the command to the specified instances.
9. (Optional) Choose **View result**.
10. (Optional) To view the command output, select a command from the list of recent commands.

11. (Optional) Choose **Output, View Output**. The image below shows a snippet of the container instance output for the **yum update** command.

**Note**

Unless you configure a command to save the output to an Amazon S3 bucket, then the command output is truncated at 2500 characters.

---

**Starting a Task at Container Instance Launch Time**

Depending on your application architecture design, you may 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. If you have not done so already, create a task definition with the container you want to run on your container instance at launch by following the procedures in Creating a Task Definition (p. 188).

2. Modify your ecsInstanceRole IAM role to add permissions for the StartTask API operation. For more information, see Amazon ECS Container Instance IAM Role (p. 415).
   b. In the navigation pane, choose Roles.
   c. Choose the ecsInstanceRole. If the role does not exist, use the procedure in Amazon ECS Container Instance IAM Role (p. 415) to create the role and return to this procedure. If the role does exist, select the role to view the attached policies.
   d. In the Permissions tab, choose Add inline policy.
   e. For Service, choose Choose a service, EC2 Container Service.
   f. For Actions, type StartTask in the search field, and then select StartTask.
   g. For Resources, select All resources, and then choose Review policy.
   h. On the Review policy page, enter a name for your policy, such as ecs-start-task and choose Create policy.

3. Launch one or more container instances using the Amazon ECS-optimized Amazon Linux 2 AMI by following the procedure in Launching an Amazon ECS Container Instance (p. 101), but in Step 6.g (p. 103) copy and paste the MIME multi-part user data script below into the User data field. Substitute 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.

   **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.

   ```bash
   #!/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
   # Install the AWS CLI and the jq JSON parser
   yum install -y aws-cli jq
   START_TASK_SCRIPT_FILE="/etc/ecs/ecs-start-task.sh"
   cat "EOF" > ${START_TASK_SCRIPT_FILE}
   exec 2>>/var/log/ecs/ecs-start-task.log
   set -x
   # Wait for the ECS service to be responsive
   until curl -s http://localhost:51678/v1/metadata
   do
     sleep 1
   done
   # Grab the container instance ARN and AWS Region from instance metadata
   instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.ContainerInstanceArn' | awk -F/ '{print $NF}' )
   API Version 2014-11-13
   129```
cluster=$(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

# 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=always
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}"

4. Verify that your container instances launch into the correct cluster and that your tasks have started.
   a. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
   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.

Each container instance you launched should have your task running on it, and the container instance ARN should be in the Started By column.
If you do not see your tasks, you can log in to your container instances with SSH and check the \texttt{/var/log/ecs/ecs-start-task.log} file for debugging information.

## Deregister a Container Instance

When you are finished with a container instance, you can 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 (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 a Classic Load Balancer or an Application Load Balancer target group are deregistered. They begin connection draining according to the settings on the load balancer or target group.

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.

**Important**

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 with the latest Amazon ECS-optimized Amazon Linux 2 AMI in the new cluster. For more information, see Terminate Your Instance in the Amazon EC2 User Guide for Linux Instances and Launching an Amazon ECS Container Instance (p. 101).

Deregistering a container instance removes the instance from a cluster, but it does not terminate the EC2 instance. If you are finished using the instance, be sure to terminate it in the Amazon EC2 console to stop billing. For more information, see Terminate Your Instance in the Amazon EC2 User Guide for Linux Instances.

**Note**

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.

**To deregister a container instance**

1. Open the Amazon ECS console at \url{https://console.aws.amazon.com/ecs/}.
2. From the navigation bar, choose the Region in which your container instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts your container instance.
4. On the **Cluster** : \texttt{name} page, choose **ECS Instances**.
5. Select the container instance ID to deregister.
6. On the **Container Instance : id** page, choose **Deregister**.
7. Review the deregistration message, and choose **Yes, 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 for Linux Instances*.

**Note**

If your instance is maintained by an Auto Scaling group or AWS CloudFormation stack, terminate the instance by updating the Auto Scaling group or AWS CloudFormation stack. Otherwise, the Auto Scaling group re-creates the instance after you terminate it.
Amazon ECS Container Agent

The Amazon ECS container agent allows container instances to connect to your cluster. The Amazon ECS container agent is included in the Amazon ECS-optimized AMIs, but you can also install it on any Amazon EC2 instance that supports the Amazon ECS specification. The Amazon ECS container agent is only supported on Amazon EC2 instances.

The source code for the Amazon ECS container agent is available on GitHub. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services does not currently support running modified copies of this software.

**Note**
The Amazon ECS container agent is installed on the AWS managed infrastructure used for tasks using the Fargate launch type. If you are only using tasks with the Fargate launch type no additional configuration is needed and the content in this topic does not apply.

**Topics**
- Installing the Amazon ECS Container Agent (p. 133)
- Amazon ECS Container Agent Versions (p. 142)
- Updating the Amazon ECS Container Agent (p. 146)
- Amazon ECS Container Agent Configuration (p. 152)
- Private Registry Authentication for Container Instances (p. 164)
- Automated Task and Image Cleanup (p. 167)
- Amazon ECS Container Metadata File (p. 168)
- Amazon ECS Task Metadata Endpoint (p. 172)
- Amazon ECS Container Agent Introspection (p. 180)
- HTTP Proxy Configuration (p. 182)

**Installing the Amazon ECS Container Agent**

If your container instance was not launched using an Amazon ECS-optimized AMI, you can install the Amazon ECS container agent manually using one of the following procedures.

- For Amazon Linux 2 instances, you can install the agent using the `amazon-linux-extras` command. For more information, see [Installing the Amazon ECS Container Agent on an Amazon Linux 2 EC2 Instance](p. 134).
- For Amazon Linux AMI instances, you can install the agent using the Amazon YUM repo. For more information, see [Installing the Amazon ECS Container Agent on an Amazon Linux AMI EC2 Instance](p. 134).
- For non-Amazon Linux instances, you can either download the agent from one of the regional S3 buckets or from Docker Hub. If you download from one of the regional S3 buckets, you can optionally verify the validity of the container agent file using the PGP signature. For more information, see [Installing the Amazon ECS Container Agent on a non-Amazon Linux EC2 Instance](p. 135).

**Note**
The Amazon ECS container agent is included in the Amazon ECS-optimized AMIs and does not require installation.
Installing the Amazon ECS Container Agent on an Amazon Linux 2 EC2 Instance

To install the Amazon ECS container agent on an Amazon Linux 2 EC2 instance using the `amazon-linux-extras` command, use the following steps.

**To install the Amazon ECS container agent on an Amazon Linux 2 EC2 instance**

1. Launch an Amazon Linux 2 EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see Amazon ECS Container Instance IAM Role (p. 415).
2. Connect to your instance.
3. Disable the `docker` Amazon Linux extra repository. The `ecs` Amazon Linux extra repository ships with its own version of Docker, so the `docker` extra must be disabled to avoid any potential future conflicts. This ensures that you are always using the Docker version that Amazon ECS intends for you to use with a particular version of the container agent.

   ```shell
   [ec2-user ~]# sudo amazon-linux-extras disable docker
   ```

4. Install and enable the `ecs` Amazon Linux extra repository.

   ```shell
   [ec2-user ~]# sudo amazon-linux-extras install -y ecs; sudo systemctl enable --now ecs
   ```

5. (Optional) You can verify that the agent is running and see some information about your new container instance with the agent introspection API. For more information, see the section called “Amazon ECS Container Agent Introspection” (p. 180).

   ```shell
   [ec2-user ~]# curl -s http://localhost:51678/v1/metadata | python -mjson.tool
   ```

   Output:

   ```json
   { 
   "Cluster": "default",
   "ContainerInstanceArn": "<container_instance_ARN>",
   "Version": "Amazon ECS Agent - v1.28.0 (d2456beb)"
   }
   ```

Installing the Amazon ECS Container Agent on an Amazon Linux AMI EC2 Instance

To install the Amazon ECS container agent on an Amazon Linux AMI EC2 instance using the Amazon YUM repo, use the following steps.

**To install the Amazon ECS container agent on an Amazon Linux AMI EC2 instance**

1. Launch an Amazon Linux AMI EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see Amazon ECS Container Instance IAM Role (p. 415).
2. Connect to your instance.
3. Install the `ecs-init` package. For more information about `ecs-init`, see the source code on GitHub.

   ```shell
   [ec2-user ~]# sudo yum install -y ecs-init
   ```
4. Start the Docker daemon.

```bash
[ec2-user ~]$ sudo service docker start
```

Output:

```
Starting cgconfig service: [ OK ]
Starting docker: [ OK ]
```

5. Start the `ecs-init` upstart job.

```bash
[ec2-user ~]$ sudo start ecs
```

Output:

```
ecs start/running, process 2804
```

6. (Optional) You can verify that the agent is running and see some information about your new container instance with the agent introspection API. For more information, see the section called “Amazon ECS Container Agent Introspection” (p. 180).

```bash
[ec2-user ~]$ curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Output:

```json
{
  "Cluster": "default",
  "ContainerInstanceArn": "<container_instance_ARN>",
  "Version": "Amazon ECS Agent - v1.28.0 (d2456beb)"
}
```

Installing the Amazon ECS Container Agent on a non-Amazon Linux EC2 Instance

To install the Amazon ECS container agent on a non-Amazon Linux EC2 instance, you can either download the agent from one of the regional S3 buckets or from Docker Hub. If you download from one of the regional S3 buckets, you can optionally verify the validity of the container agent file using the PGP signature.

The latest Amazon ECS container agent files, by region, are listed below for reference.

<table>
<thead>
<tr>
<th>Region</th>
<th>Region Name</th>
<th>Container agent</th>
<th>Container agent signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>US East (Ohio)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>us-east-1</td>
<td>US East (N. Virginia)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>us-west-1</td>
<td>US West (N. California)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>us-west-2</td>
<td>US West (Oregon)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>Region</td>
<td>Region Name</td>
<td>Container agent</td>
<td>Container agent signature</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------</td>
<td>------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>Asia Pacific (Hong Kong)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>Asia Pacific (Tokyo)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>Asia Pacific (Seoul)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>Asia Pacific (Mumbai)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>Asia Pacific (Singapore)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>Asia Pacific (Sydney)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>Canada (Central)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>EU (Frankfurt)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>EU (Ireland)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>EU (London)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>EU (Paris)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>South America (São Paulo)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>China (Beijing)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>China (Ningxia)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>AWS GovCloud (US-East)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
<tr>
<td>us-gov-west-1</td>
<td>AWS GovCloud (US-West)</td>
<td>ECS container agent</td>
<td>PGP signature</td>
</tr>
</tbody>
</table>

**To install the Amazon ECS container agent on a non-Amazon Linux EC2 instance**

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 (p. 415).
2. Connect to your instance.
3. Install Docker on your instance. Amazon ECS requires a minimum Docker version of 1.9.0 (version 18.06.1-ce is recommended), and the default Docker versions in many system package managers, such as `yum` or `apt-get` do not meet this minimum requirement. For information about installing the latest Docker version on your particular Linux distribution, see https://docs.docker.com/engine/installation/.

   **Note**
   
   The Amazon Linux AMI always includes the recommended version of Docker for use with Amazon ECS. You can install Docker on Amazon Linux with the `sudo yum install docker -y` command.

4. Check your Docker version to verify that your system meets the minimum version requirement.

   ```bash
   ubuntu:~$ sudo docker version
   ```
Output:

Client version: 1.4.1
Client API version: 1.16
Go version (client): go1.3.3
Git commit (client): Sbc2ff8
OS/Arch (client): linux/amd64
Server version: 1.4.1
Server API version: 1.16
Go version (server): go1.3.3
Git commit (server): Sbc2ff8

In this example, the Docker version is 1.4.1, which is below the minimum version of 1.9.0. This instance needs to upgrade its Docker version before proceeding. For information about installing the latest Docker version on your particular Linux distribution, go to https://docs.docker.com/engine/installation/.

5. Run the following commands on your container instance to allow the port proxy to route traffic using loopback addresses.

```
ubuntu:~$ sudo sh -c "echo 'net.ipv4.conf.all.route_localnet = 1' >> /etc/sysctl.conf"
ubuntu:~$ sudo sysctl -p /etc/sysctl.conf
```

6. Run the following commands on your container instance to enable IAM roles for tasks. For more information, see IAM Roles for Tasks (p. 435).

```
ubuntu:~$ sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-destination 127.0.0.1:51679
ubuntu:~$ sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp --dport 80 -j REDIRECT --to-ports 51679
```

7. Write the new `iptables` configuration to your operating system-specific location.

   • For Debian/Ubuntu:

     ```
sudo sh -c 'iptables-save > /etc/iptables/rules.v4'
     ```

   • For CentOS/RHEL:

     ```
sudo sh -c 'iptables-save > /etc/sysconfig/iptables'
     ```

8. Create the `/etc/ecs` directory and create the Amazon ECS container agent configuration file.

```
ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
```

9. Edit the `/etc/ecs/ecs.config` file and add the following contents. 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
```
Installing the Amazon ECS Container Agent on a non-Amazon Linux EC2 Instance

For more information about these and other agent runtime options, see Amazon ECS Container Agent Configuration (p. 152).

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 Container Instance Configuration in Amazon S3 (p. 163) and Private Registry Authentication for Tasks (p. 251).

10. Pull and 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. For more information, see Automatically start containers and Restart policies in the Docker documentation. The Linux variants of the Amazon ECS-optimized AMI use the ecs-init RPM for this purpose, and you can view the source code for this RPM on GitHub. For example systemd unit files for Ubuntu 16.04 and CentOS 7, see Example Container Instance User Data Configuration Scripts (p. 108).

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 (p. 152).

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.

a. (Optional) Download the ECS container agent tarball from the regional S3 URL and load it. If you don't download the agent tarball from S3, the docker run command in the next step will download it from Docker Hub for you automatically.


Note
To download other versions of the Amazon ECS container agent, use one of the following formats, changing the version number in the URL:

ecs-agent-<version>.tar
ecs-agent-<SHA>.tar

For example:
Load the ECS container agent image.

```
ubuntu:~$ sudo docker load --input ./ecs-agent.tar
```

b. Run the ECS container agent image.

```
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 --net=host --env-file=/etc/ecs/ecs.config --net=host --env-file=/etc/ecs/ecs.config
amazon/amazon-ecs-agent:latest
```

**Important**
The host network mode is the only supported network mode for the container agent container. For more information, see [Running the Amazon ECS Container Agent with Host Network Mode](p. 142).

**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.

11. (Optional) If you downloaded the Amazon ECS container agent file from S3, you can verify the validity of the file.

a. Download and install GnuPG. For more information about GnuPG, see the [GnuPG website](http://www.gnupg.org). For Linux systems, install gpg using the package manager on your flavor of Linux.

b. Retrieve the Amazon ECS PGP public key. You can use a command to do this or manually create the key and then import it.

i. Option 1: Retrieve the key with the following command.

```
  gpg --keyserver hkp://keys.gnupg.net --recv BCE9D9A42D51784F
```

ii. Option 2: Create a file with the following contents of the Amazon ECS PGP public key and then import it:

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

mQINBFq1SasBEADl1Gct1NVJ1ydIn8DqehYYe9ne3dt6jqKFmKowLmm6LLGJe7HU j0tqgCWDRkN+qPhHdqArsGdZAt2pXY5fEiHpHgar5CP8QgRNR02fl741mavr4Vg 7K/KH8Yh12qRw32/B94LEgRbbGTDmWFWdXuxoPctt8QaMj3L3G6Pe+6xVRWkChQ Bu0AhjBQ+bEm0kNyoJLJnJ1L3UMAG56t8E3LANIgGmnpN8B1uwJwUpOZoTQX N+6pEBjRJKL/1/ETU4FxYwzwvWNahxeNRnoYj3uyvKekiciRwy4kjo+ski2BqO 2K7oVXhOc3j6+3l1lH/qDLxMCb2az5MM1mOoF8EXS5HmNq1KFw5xgX6N1iC0 1FRfr7Q0d5fMN1d3FanLGy/zn1rsSaqj0L6zrsq8O4HLN0WBvBdehk2Kr+5kFxn 5LbpzGpFGpFr5JhQ+kTHMA9fZ87yUc64BjN6FF91fJuSUfQbdrkvRLqRqBcBG9Qx3 rJAEhieJvMUNs1+1EgeCrxj5xuSkNWTlzwcj3hQ2qBr4ADLV+hvFJkGz2g9m6Z3bxq 1Tn0WcX4rXwUEBA2gX+2LdHeVd78a3gtEaSSTfQoO8sYXaqbvl0sW0ooc1y/5Zb zisHTJIlLtUy8s9WisiSp2os0emeHZ1cVMFw61EgPrJAIupgc7ky2VFt4YwfwARAQB

```

API Version 2014-11-13

139
Installing the Amazon ECS Container Agent on a non-Amazon Linux EC2 Instance

API Version 2014-11-13
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 A6C6 B09E 5D48

Import the Amazon ECS PGP public key with the following command.

```
aws ecr get-public-policy --repository-name my-repository
```
Running the Amazon ECS Container 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 enables 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 can not access IAM role credentials from the container instance profile and enforces that tasks use only the IAM task role credentials. For more information, see IAM Roles for Tasks (p. 435).

This also makes it so the container agent doesn’t contend for connections and network traffic on the docker0 bridge.

Amazon ECS Container Agent Versions

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 (p. 146).

Launching your container instances from the most recent Amazon ECS-optimized Amazon Linux 2 AMI ensures that 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 Container Instance (p. 101).

To install the latest version of the Amazon ECS container agent on another operating system, see Installing the Amazon ECS Container Agent (p. 133). The table in Amazon ECS-Optimized Amazon Linux 2 AMI Versions (p. 82) shows the Docker version that is tested on Amazon Linux 2 for each agent version. The table in Amazon ECS-Optimized Amazon Linux AMI Container Agent Versions (p. 144) shows the Docker version that is tested on the Amazon Linux AMI for each agent version.

To see which features and enhancements are included with each agent release, see https://github.com/aws/amazon-ecs-agent/releases.

Amazon ECS-Optimized Amazon Linux 2 AMI Container Agent Versions

The Amazon ECS-optimized Amazon Linux 2 AMI comes prepackaged with the Amazon ECS container agent, Docker, and the `ecs-init systemd` service that controls the starting and stopping of the agent at boot and shutdown. The following table lists the container agent version, the `ecs-init` version, and the Docker version that is tested and packaged with each Amazon ECS-optimized Amazon Linux 2 AMI.

**Note**
As new Amazon ECS-optimized Amazon Linux 2 AMIs and Amazon ECS agent versions are released, older versions are still available for launch in Amazon EC2. However, we encourage you to update to the latest version (p. 146) of the Amazon ECS agent and to keep your container instance software up-to-date. If you request support for an older version of the Amazon ECS agent through AWS Support, you may be asked to move to the latest version as a part of the support process.

**Important**
Amazon ECS agent versions 1.20.0 and later have deprecated support for Docker versions older than 1.9.0.

<table>
<thead>
<tr>
<th>Amazon ECS-optimized Amazon Linux 2 AMI</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
<th><code>ecs-init</code> version</th>
</tr>
</thead>
<tbody>
<tr>
<td>20190402</td>
<td>1.27.0</td>
<td>18.06.1-ce</td>
<td>1.27.0-1</td>
</tr>
<tr>
<td>20190301</td>
<td>1.26.0</td>
<td>18.06.1-ce</td>
<td>1.26.0-1</td>
</tr>
<tr>
<td>20190215</td>
<td>1.25.3</td>
<td>18.06.1-ce</td>
<td>1.25.3-1</td>
</tr>
<tr>
<td>20190204</td>
<td>1.25.2</td>
<td>18.06.1-ce</td>
<td>1.25.2-1</td>
</tr>
<tr>
<td>20190127</td>
<td>1.25.1</td>
<td>18.06.1-ce</td>
<td>1.25.1-1</td>
</tr>
<tr>
<td>20190118</td>
<td>1.25.0</td>
<td>18.06.1-ce</td>
<td>1.25.0-1</td>
</tr>
<tr>
<td>20190107</td>
<td>1.24.0</td>
<td>18.06.1-ce</td>
<td>1.24.0-1</td>
</tr>
<tr>
<td>20181112</td>
<td>1.22.0</td>
<td>18.06.1-ce</td>
<td>1.22.0-1</td>
</tr>
<tr>
<td>20181016</td>
<td>1.20.3</td>
<td>18.06.1-ce</td>
<td>1.21.0-1</td>
</tr>
</tbody>
</table>

For more information about the Amazon ECS-optimized Amazon Linux 2 AMI, including AMI IDs for the latest version in each region, see Amazon ECS-optimized AMIs (p. 62).
Amazon ECS-Optimized Amazon Linux AMI Container Agent Versions

The Amazon ECS-optimized Amazon Linux AMI comes prepackaged with the Amazon ECS container agent, Docker, and the `ecs-init` service that controls the starting and stopping of the agent at boot and shutdown. The following table lists the container agent version, the `ecs-init` version, and the Docker version that is tested and packaged with each Amazon ECS-optimized AMI.

**Note**
As new Amazon ECS-optimized Amazon Linux AMIs and Amazon ECS agent versions are released, older versions are still available for launch in Amazon EC2. However, we encourage you to update to the latest version (p. 146) of the Amazon ECS agent and to keep your container instance software up-to-date. If you request support for an older version of the Amazon ECS agent through AWS Support, you may be asked to move to the latest version as a part of the support process.

**Important**
Amazon ECS agent versions 1.20.0 and later have deprecated support for Docker versions older than 1.9.0.

<table>
<thead>
<tr>
<th>Amazon ECS-optimized Amazon Linux AMI</th>
<th>Amazon ECS container agent version</th>
<th>Docker version</th>
<th><code>ecs-init</code> version</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.03.p</td>
<td>1.27.0</td>
<td>18.06.1-ce</td>
<td>1.27.0-1</td>
</tr>
<tr>
<td>2018.03.o</td>
<td>1.26.0</td>
<td>18.06.1-ce</td>
<td>1.26.0-1</td>
</tr>
<tr>
<td>2018.03.n</td>
<td>1.25.3</td>
<td>18.06.1-ce</td>
<td>1.25.3-1</td>
</tr>
<tr>
<td>2018.03.m</td>
<td>1.25.2</td>
<td>18.06.1-ce</td>
<td>1.25.2-1</td>
</tr>
<tr>
<td>2018.03.l</td>
<td>1.25.1</td>
<td>18.06.1-ce</td>
<td>1.25.1-1</td>
</tr>
<tr>
<td>2018.03.k</td>
<td>1.25.0</td>
<td>18.06.1-ce</td>
<td>1.25.0-1</td>
</tr>
<tr>
<td>2018.03.j</td>
<td>1.24.0</td>
<td>18.06.1-ce</td>
<td>1.24.0-1</td>
</tr>
<tr>
<td>2018.03.i</td>
<td>1.22.0</td>
<td>18.06.1-ce</td>
<td>1.22.0-1</td>
</tr>
<tr>
<td>2018.03.h</td>
<td>1.21.0</td>
<td>18.06.1-ce</td>
<td>1.21.0-1</td>
</tr>
<tr>
<td>2018.03.g</td>
<td>1.20.3</td>
<td>18.06.1-ce</td>
<td>1.20.3-1</td>
</tr>
<tr>
<td>2018.03.f</td>
<td>1.20.2</td>
<td>18.06.1-ce</td>
<td>1.20.2-1</td>
</tr>
<tr>
<td>2018.03.e</td>
<td>1.20.1</td>
<td>18.03.1-ce</td>
<td>1.20.1-1</td>
</tr>
<tr>
<td>2018.03.d</td>
<td>1.20.0</td>
<td>18.03.1-ce</td>
<td>1.20.0-1</td>
</tr>
<tr>
<td>2018.03.c</td>
<td>1.19.1</td>
<td>18.03.1-ce</td>
<td>1.19.1-1</td>
</tr>
<tr>
<td>2018.03.b</td>
<td>1.19.0</td>
<td>18.03.1-ce</td>
<td>1.19.0-1</td>
</tr>
<tr>
<td>2018.03.a</td>
<td>1.18.0</td>
<td>17.12.1-ce</td>
<td>1.18.0-1</td>
</tr>
<tr>
<td>2017.09.l</td>
<td>1.17.3</td>
<td>17.12.1-ce</td>
<td>1.17.3-1</td>
</tr>
<tr>
<td>2017.09.k</td>
<td>1.17.2</td>
<td>17.12.0-ce</td>
<td>1.17.2-1</td>
</tr>
<tr>
<td>Amazon ECS-optimized Amazon Linux AMI</td>
<td>Amazon ECS container agent version</td>
<td>Docker version</td>
<td>ecs-init version</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------------------------------</td>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>2017.09.j</td>
<td>1.17.2</td>
<td>17.12.0-ce</td>
<td>1.17.2-1</td>
</tr>
<tr>
<td>2017.09.i</td>
<td>1.17.1</td>
<td>17.09.1-ce</td>
<td>1.17.1-1</td>
</tr>
<tr>
<td>2017.09.h</td>
<td>1.17.0</td>
<td>17.09.1-ce</td>
<td>1.17.0-2</td>
</tr>
<tr>
<td>2017.09.g</td>
<td>1.16.2</td>
<td>17.09.1-ce</td>
<td>1.16.2-1</td>
</tr>
<tr>
<td>2017.09.f</td>
<td>1.16.1</td>
<td>17.06.2-ce</td>
<td>1.16.1-1</td>
</tr>
<tr>
<td>2017.09.e</td>
<td>1.16.1</td>
<td>17.06.2-ce</td>
<td>1.16.1-1</td>
</tr>
<tr>
<td>2017.09.d</td>
<td>1.16.0</td>
<td>17.06.2-ce</td>
<td>1.16.0-1</td>
</tr>
<tr>
<td>2017.09.c</td>
<td>1.15.2</td>
<td>17.06.2-ce</td>
<td>1.15.1-1</td>
</tr>
<tr>
<td>2017.09.b</td>
<td>1.15.1</td>
<td>17.06.2-ce</td>
<td>1.15.1-1</td>
</tr>
<tr>
<td>2017.09.a</td>
<td>1.15.0</td>
<td>17.06.2-ce</td>
<td>1.15.0-4</td>
</tr>
<tr>
<td>2017.03.g</td>
<td>1.14.5</td>
<td>17.03.2-ce</td>
<td>1.14.5-1</td>
</tr>
<tr>
<td>2017.03.f</td>
<td>1.14.4</td>
<td>17.03.2-ce</td>
<td>1.14.4-1</td>
</tr>
<tr>
<td>2017.03.e</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2017.03.d</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2017.03.c</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2017.03.b</td>
<td>1.14.3</td>
<td>17.03.1-ce</td>
<td>1.14.3-1</td>
</tr>
<tr>
<td>2016.09.g</td>
<td>1.14.1</td>
<td>1.12.6</td>
<td>1.14.1-1</td>
</tr>
<tr>
<td>2016.09.e</td>
<td>1.14.0</td>
<td>1.12.6</td>
<td>1.14.0-1</td>
</tr>
<tr>
<td>2016.09.c</td>
<td>1.13.1</td>
<td>1.11.2</td>
<td>1.13.1-1</td>
</tr>
<tr>
<td>2016.09.b</td>
<td>1.13.1</td>
<td>1.11.2</td>
<td>1.13.1-1</td>
</tr>
<tr>
<td>2016.09.a</td>
<td>1.13.0</td>
<td>1.11.2</td>
<td>1.13.0-1</td>
</tr>
<tr>
<td>2016.03.j</td>
<td>1.13.0</td>
<td>1.11.2</td>
<td>1.13.0-1</td>
</tr>
<tr>
<td>2016.03.i</td>
<td>1.12.2</td>
<td>1.11.2</td>
<td>1.12.2-1</td>
</tr>
<tr>
<td>2016.03.h</td>
<td>1.12.1</td>
<td>1.11.2</td>
<td>1.12.1-1</td>
</tr>
<tr>
<td>2016.03.g</td>
<td>1.12.0</td>
<td>1.11.2</td>
<td>1.12.0-1</td>
</tr>
<tr>
<td>2016.03.f</td>
<td>1.11.1</td>
<td>1.11.2</td>
<td>1.11.1-1</td>
</tr>
<tr>
<td>2016.03.e</td>
<td>1.11.0</td>
<td>1.11.2</td>
<td>1.11.0-1</td>
</tr>
<tr>
<td>2016.03.d</td>
<td>1.10.0</td>
<td>1.11.1</td>
<td>1.10.0-1</td>
</tr>
</tbody>
</table>
For more information about the Amazon ECS-optimized Amazon Linux AMI, including AMI IDs for the latest version in each region, see Amazon ECS-optimized AMIs (p. 62).

### Updating the Amazon ECS Container Agent

Occasionally, you may 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.

**Topics**

- Checking Your Amazon ECS Container Agent Version (p. 147)
- Updating the Amazon ECS Container Agent on an Amazon ECS-optimized AMI (p. 148)
- Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-Optimized AMIs) (p. 150)
Checking Your 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.

To check if your Amazon ECS container agent is running the latest version in the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the Clusters page, select the cluster that hosts the container instance or instances to check.
3. On the Cluster: cluster_name page, choose ECS Instances.
4. Note the Agent version column for your container instances. If you are using an outdated agent version on any of your container instances, the console alerts you with a message and flags the outdated agent version.

If your agent version is 1.28.0, you are running the latest container agent. If your agent version is below 1.28.0, 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 (p. 148).
- 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) (p. 150).

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 Container Instance (p. 101).

You can also use the Amazon ECS container agent introspection API to check the agent version from the container instance itself. For more information, see Amazon ECS Container Agent Introspection (p. 180).

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 | python -mjson.tool
```

API Version 2014-11-13

147
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 your current container instances 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 AMIs (p. 62).

- 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 (p. 148).

- 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 (p. 149).

  **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. For more information, see Connect to Your Container Instance (p. 113).

2. Update the ecs-init package with the following command.

   ```bash
   [ec2-user ~]$ 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**

This update process is only supported on Linux variants of the Amazon ECS-optimized AMI. For container instances that are running other operating systems, see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-Optimized AMIs) (p. 150).

**Note**

Agent updates with the UpdateContainerAgent API operation 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 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 Container Instance (p. 101).

The update 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, the update process progresses as shown in the flow chart below. If an update is not available, for example, if the agent is already running the most recent version, then a NoUpdateAvailableException is returned.

![Flow Chart](chart.png)

The stages in the update process shown above are as follows:
An agent update is available, and the update process has started.

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.

The agent download has completed and the agent contents have been verified.

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.

**To update the Amazon ECS container agent on an Amazon ECS-optimized AMI in the console**

*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. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the Clusters page, select the cluster that hosts the container instance or instances to check.
3. On the Cluster: `cluster_name` page, choose ECS Instances.
4. Select the container instance to update.
5. On the Container Instance page, choose Update agent.

**To update the Amazon ECS container agent on an Amazon ECS-optimized AMI with the AWS CLI**

*Note*
Agent updates with the UpdateContainerAgent API operation do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

- Use the following command to update the Amazon ECS container agent on your container instance:

```
aws ecs update-container-agent --cluster cluster_name --container-instance container_instance_id
```

**Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-Optimized AMIs)**

**To manually update the Amazon ECS container agent (for non-Amazon ECS-optimized AMIs)**

*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.

```bash
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.

```bash
ubuntu:~$ docker stop ecs-agent
```

4. Delete the agent container.

```bash
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`.

```bash
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
ECSAVAILABLE_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 (p. 152).

**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 Container Instance Configuration in Amazon S3 (p. 163) and Private Registry Authentication for Tasks (p. 251).

7. Pull the latest Amazon ECS container agent image from Docker Hub.

```bash
ubuntu:~$ docker pull amazon/amazon-ecs-agent:latest
```

Output:

```
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
```
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. For more information, see [Automatically start containers and Restart policies](https://docs.docker.com/engine/userguide/operating-systems/daemonizing/) in the Docker documentation. 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. For example, `systemd` unit files for Ubuntu 16.04 and CentOS 7, see [Example Container Instance User Data Configuration Scripts](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/container-instance-user-data-configuration.html) (p. 108).

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](p. 152).

**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.

  ```bash
  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.

  ```bash
  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
  --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 Container Agent Configuration

The Amazon ECS container agent supports a number of configuration options, most of which should be set through environment variables. The following environment variables are available, and all of them are optional.
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 Container Instances with Amazon EC2 User Data (p. 105).

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 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.

### Available Parameters

The following are the available environment keys:

**ECS_CLUSTER**

Example values: `MyCluster`

Default value on Linux: `default`

Default value on Windows: `default`

The cluster that this agent should check into. If this value is undefined, then the `default` cluster is assumed. If the `default` cluster does not exist, the Amazon ECS container agent attempts to create it. If a non-`default` cluster is specified and it does not exist, registration fails.

**ECS_RESERVED_PORTS**

Example values: `[22, 80, 5000, 8080]`

Default value on Linux: `[22, 2375, 2376, 51678, 51679, 51680]`

Default value on Windows: `[53, 135, 139, 445, 2375, 2376, 3389, 5985, 51678, 51679]`

An array of ports that should be marked as unavailable for scheduling on this container instance.

**ECS_RESERVED_PORTS_UDP**

Example values: `[53, 123]`

Default value on Linux: `[]`

Default value on Windows: `[]`

An array of UDP ports that should be marked as unavailable for scheduling on this container instance.

**ECS_ENGINE_AUTH_TYPE**

Example values: `dockercfg` | `docker`

Default value on Linux: `Null`

Default value on Windows: `Null`

Required for private registry authentication. This is the type of authentication data in `ECS_ENGINE_AUTH_DATA`. For more information, see Authentication Formats (p. 164).
ECS_ENGINE_AUTH_DATA

Example values:

- ECS_ENGINE_AUTH_TYPE=dockercfg:
  
  ```json
  "https://index.docker.io/v1/":
  
  
  
  "auth": "zq212MzEXAMPLE7o6T25Dk0i",
  "email": "email@example.com"
  ```

- ECS_ENGINE_AUTH_TYPE=docker:
  
  ```json
  "https://index.docker.io/v1/":
  
  
  
  "username": "my_name",
  "password": "my_password",
  "email": "email@example.com"
  ```

Default value on Linux: Null
Default value on Windows: Null

Required for private registry authentication. If ECS_ENGINE_AUTH_TYPE=dockercfg, then the ECS_ENGINE_AUTH_DATA value should be the contents of a Docker configuration file (~/.dockercfg or ~/.docker/config.json) created by running docker login. If ECS_ENGINE_AUTH_TYPE=docker, then the ECS_ENGINE_AUTH_DATA value should be a JSON representation of the registry server to authenticate against, as well as the authentication parameters required by that registry (such as user name, password, and email address for that account). For more information, see Authentication Formats (p. 164).

AWS_DEFAULT_REGION

Example values: us-east-1
Default value on Linux: Taken from EC2 instance metadata.
Default value on Windows: Taken from EC2 instance metadata.

The region to be used in API requests as well as to infer the correct backend host.

AWS_ACCESS_KEY_ID

Example values: AKIAIOSFODNN7EXAMPLE
Default value on Linux: Taken from EC2 instance metadata.
Default value on Windows: Taken from EC2 instance metadata.

The access key used by the agent for all calls.

AWS_SECRET_ACCESS_KEY

Example values: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default value on Linux: Taken from EC2 instance metadata.
Default value on Windows: Taken from EC2 instance metadata.

The secret key used by the agent for all calls.

AWS_SESSION_TOKEN

Default value on Linux: Taken from EC2 instance metadata.
Default value on Windows: Taken from EC2 instance metadata.

The session token used for temporary credentials.

DOCKER_HOST

Example values: unix:///var/run/docker.sock
Default value on Linux: unix:///var/run/docker.sock
Default value on Windows: npipe:///./pipe/docker_engine
Used to create a connection to the Docker daemon; behaves similarly to the environment variable as used by the Docker client.

**ECS_LOGLEVEL**

Example values: crit | error | warn | info | debug

Default value on Linux: info

Default value on Windows: info

The level to log at on stdout.

**ECS_LOGFILE**

Example values: /ecs-agent.log

Default value on Linux: Null

Default value on Windows: Null

The path to output full debugging information to. If blank, no logs are recorded. If this value is set, it logs at the debug level (regardless of ECS_LOGLEVEL) are written to that file.

**ECS_CHECKPOINT**

Example values: true | false

Default value on Linux: If ECS_DATADIR is explicitly set to a non-empty value, then ECS_CHECKPOINT is set to true; otherwise, it is set to false.

Default value on Windows: If ECS_DATADIR is explicitly set to a non-empty value, then ECS_CHECKPOINT is set to true; otherwise, it is set to false.

Whether to save the checkpoint state to the location specified with ECS_DATADIR.

**ECS_DATADIR**

Example values: /data

Default value on Linux: /data/

Default value on Windows: C:\ProgramData\Amazon\ECS\data

The name of the persistent data directory on the container that is running the Amazon ECS container agent. The directory is used to save information about the cluster and the agent state.

**ECS_UPDATES_ENABLED**

Example values: true | false

Default value on Linux: false

Default value on Windows: false

Whether to exit for ECS agent updates when they are requested.

**ECS_UPDATE_DOWNLOAD_DIR**

Example values: /cache

Default value on Linux: Null

Default value on Windows: Null

The filesystem location to place update tarballs within the container when they are downloaded.

**ECS_DISABLE_METRICS**

Example values: true | false
Available Parameters

Default value on Linux: false
Default value on Windows: true

Whether to disable CloudWatch metrics for Amazon ECS. If this value is set to true, CloudWatch metrics are not collected.

**ECS_POLL_METRICS**

Example values: true | false
Default value on Linux: false
Default value on Windows: false

Whether to poll or stream when gathering CloudWatch metrics for tasks.

**ECS_POLLING_METRICS_WAIT_DURATION**

Example values: 30s
Default value on Linux: 15s
Default value on Windows: 15s

Time to wait to poll for new CloudWatch metrics for a task. Only used when ECS_POLL_METRICS is true.

**ECS_RESERVED_MEMORY**

Example values: 32
Default value on Linux: 0
Default value on Windows: 0

The amount of memory, in MiB, to remove from the pool that is allocated to your tasks. This effectively reserves that memory for critical system processes including the Docker daemon and the Amazon ECS container agent. For example, if you specify ECS_RESERVED_MEMORY=256, then the agent registers the total memory minus 256 MiB for that instance, and 256 MiB of the system memory cannot be allocated by ECS tasks. For more information, see Container Instance Memory Management (p. 124).

**ECS_AVAILABLE_LOGGING_DRIVERS**

Example values: ["awslogs","fluentd","gelf","json-file","journald","splunk","logentries","syslog"]
Default value on Linux: ["json-file","none"]
Default value on Windows: ["json-file","none"]

**Note**

If you are using ECS init, the default values are ["json-file","syslog","awslogs","none"].

The logging drivers available on the container instance. The Amazon ECS container agent running on a container instance must register the logging drivers available on that instance with the ECS_AVAILABLE_LOGGING_DRIVERS environment variable before containers placed on that instance can use log configuration options for those drivers in tasks. For information about how to use the awslogs log driver, see Using the awslogs Log Driver (p. 245). For more information about the different log drivers available for your Docker version and how to configure them, see Configure logging drivers in the Docker documentation.

**ECS_DISABLE_PRIVILEGED**

Example values: true | false
Default value on Linux: false
Default value on Windows: false

Whether launching privileged containers is disabled on the container instance. If this value is set to true, privileged containers are not permitted.

**ECS_SELINUX_CAPABLE**

Example values: true | false
Default value on Linux: false
Default value on Windows: false

Whether SELinux is available on the container instance.

**ECS_APPARMOR_CAPABLE**

Example values: true | false
Default value on Linux: false
Default value on Windows: false

Whether AppArmor is available on the container instance.

**ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION**

Example values: 1h (Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", and "h".)
Default value on Linux: 3h
Default value on Windows: 3h

Time duration to wait from when a task is stopped until the Docker container is removed. As this removes the Docker container data, be aware that if this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. The minimum duration is 1m; any value shorter than 1 minute is ignored.

**ECS_CONTAINER_STOP_TIMEOUT**

Example values: 10m (Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", and "h".)
Default value on Linux: 30s
Default value on Windows: 30s

Time duration to wait from when a task is stopped before its containers are forcefully killed if they do not exit normally on their own.

**ECS_CONTAINER_START_TIMEOUT**

Example values: 10m (Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", and "h".)
Default value on Linux: 3m
Default value on Windows: 8m

Time duration to wait before giving up on starting a container.

**HTTP_PROXY**

Example values: 10.0.0.131:3128
Default value on Linux: Null
Default value on Windows: Null
The hostname (or IP address) and port number of an HTTP proxy to use for the ECS agent to connect to the internet (for example, if your container instances do not have external network access through an Amazon VPC internet gateway or NAT gateway or instance). If this variable is set, you must also set the NO_PROXY variable to filter EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see HTTP Proxy Configuration (p. 182).

**NO_PROXY**

Example values:
- Linux: 169.254.169.254,169.254.170.2,/var/run/docker.sock
- Windows: 169.254.169.254,169.254.170.2,\\pipe\docker_engine

Default value on Linux: Null

Default value on Windows: Null

The HTTP traffic that should not be forwarded to the specified HTTP_PROXY. You must specify 169.254.169.254,/var/run/docker.sock to filter EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see HTTP Proxy Configuration (p. 182).

**ECS_ENABLE_TASK_IAM_ROLE**

Example values: true | false

Default value on Linux: false

Default value on Windows: false

*Note*

If you are using ECS init, the default value is true.

Whether IAM roles for tasks should be enabled on the container instance for task containers with the bridge or default network modes. For more information, see IAM Roles for Tasks (p. 435).

**ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST**

Example values: true | false

Default value on Linux: false

Default value on Windows: false

*Note*

If you are using ECS init, the default value is true.

Whether IAM roles for tasks should be enabled on the container instance for task containers with the host network mode. This variable is only supported on agent versions 1.12.0 and later. For more information, see IAM Roles for Tasks (p. 435).

**ECS_DISABLE_IMAGE_CLEANUP**

Example values: true

Default value on Linux: false

Default value on Windows: false

Whether to disable automated image cleanup for the Amazon ECS agent. For more information, see Automated Task and Image Cleanup (p. 167).

**ECS_IMAGE_CLEANUP_INTERVAL**

Example values: 30m

Default value on Linux: 30m

Default value on Windows: 30m
Available Parameters

The time interval between automated image cleanup cycles. If set to less than 10 minutes, the value is ignored.

**ECS_IMAGE_MINIMUM_CLEANUP_AGE**

Example values: 30m
Default value on Linux: 1h
Default value on Windows: 1h

The minimum time interval between when an image is pulled and when it can be considered for automated image cleanup.

**ECS_NUM_IMAGES_DELETE_PER_CYCLE**

Example values: 5
Default value on Linux: 5
Default value on Windows: 5

The maximum number of images to delete in a single automated image cleanup cycle. If set to less than 1, the value is ignored.

**ECS_IMAGE_PULL_BEHAVIOR**

Example values: default | always | once | prefer-cached

Default value on Linux: default
Default value on Windows: default

The behavior used to customize the pull image process for your container instances. The following describes the optional behaviors:

- If `default` is specified, the image is pulled remotely. If the image pull fails, then the container uses the cached image on the instance.
- If `always` is specified, the image is always pulled remotely. If the image pull fails, then the task fails. This option ensures that the latest version of the image is always pulled. Any cached images are ignored and are subject to the automated image cleanup process.
- If `once` is specified, the image is pulled remotely only if it has not been 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.
- If `prefer-cached` is specified, the image is pulled remotely if there is no cached image. Otherwise, the cached image on the instance is used. Automated image cleanup is disabled for the container to ensure that the cached image is not removed.

**ECS_IMAGE_PULL_INACTIVITY_TIMEOUT**

Example values: 1m
Default value on Linux: 1m
Default value on Windows: 3m

The time to wait after docker pulls complete waiting for extraction of a container. Useful for tuning large Windows containers.

**ECS_INSTANCE_ATTRIBUTES**

Example values: {"custom_attribute": "custom_attribute_value"}
Default value on Linux: Null
Available Parameters

Default value on Windows: Null

A list of custom attributes, in JSON form, to apply to your container instances. Using this attribute at instance registration adds the custom attributes, allowing you to skip the manual method of adding custom attributes via the AWS Management Console.

**Note**

Attributes added do not apply to container instances that are already registered. To add custom attributes to already registered container instances, see Adding an Attribute (p. 277).

For information about custom attributes to use, see Attributes (p. 276).

An invalid JSON value for this variable causes the agent to exit with a code of 5. A message appears in the agent logs. The JSON value may be valid but there is an issue detected when validating the attribute (for example if the value is too long or contains invalid characters). In that case, the container instance registration happens but the agent exits with a code of 5 and a message is written to the agent logs. For information about how to locate the agent logs, see Amazon ECS Container Agent Log (p. 600).

**ECS_ENABLE_TASK_ENI**

Example values: true | false

Default value on Linux: false

Default value on Windows: Not applicable

Whether to enable task networking for tasks to be launched with their own network interface.

**ECS_CNI_PLUGINS_PATH**

Example values: /ecs/cni

Default value on Linux: /amazon-ecs-cni-plugins

Default value on Windows: Not applicable

The path where the cni binary file is located.

**ECS_AWSVPC_BLOCK_IMDS**

Example values: true | false

Default value on Linux: false

Default value on Windows: Not applicable

Whether to block access to Instance Metadata for tasks started with awsvpc network mode.

**ECS_AWSVPC_ADDITIONAL_LOCAL_ROUTES**

Example values: ["10.0.15.0/24"]

Default value on Linux: [ ]

Default value on Windows: Not applicable

In awsvpc network mode, traffic to these prefixes is routed via the host bridge instead of the task elastic network interface.

**ECS_ENABLE_CONTAINER_METADATA**

Example values: true | false

Default value on Linux: false
Default value on Windows: false

When true, the agent creates a file describing the container's metadata. The file can be located and consumed by using the container environment variable $ECS_CONTAINER_METADATA_FILE.

**ECS_HOST_DATA_DIR**

Example values: /var/lib/ecs

Default value on Linux: /var/lib/ecs

Default value on Windows: Not applicable

The source directory on the host from which ECS_DATADIR is mounted. We use this to determine the source mount path for container metadata files in the case the ECS agent is running as a container. We do not use this value in Windows because the ECS agent does not run as a container.

**ECS_ENABLE_TASK_CPU_MEM_LIMIT**

Example values: true | false

Default value on Linux: true

Default value on Windows: false

Whether to enable task-level CPU and memory limits.

**ECS_CGROUP_PATH**

Example values: /sys/fs/cgroup

Default value on Linux: /sys/fs/cgroup

Default value on Windows: Not applicable

The root cgroup path that is expected by the ECS agent. This is the path that accessible from the agent mount.

**ECS_ENABLE_CPU_UNBOUNDED_WINDOWS_WORKAROUND**

Example values: true | false

Default value on Linux: Not applicable

Default value on Windows: false

When true, ECS allows CPU-unbounded (CPU=0) tasks to run along with CPU-bounded tasks in Windows.

**ECS_TASK_METADATA_RPS_LIMIT**

Example values: 100,150

Default value on Linux: 40,60

Default value on Windows: 40,60

Comma-separated integer values for steady state and burst throttle limits for the task metadata endpoint.

**ECS_SHARED_VOLUME_MATCH_FULL_CONFIG**

Example values: true | false

Default value on Linux: false

Default value on Windows: false
When `dockerVolumeConfiguration` is specified in a task definition and the `autopapproxion` flag is used, the Amazon ECS container agent compares the details of the Docker volume with the details of existing Docker volumes. When `ECS_SHARED_VOLUME_MATCH_FULL_CONFIG` is true, the container agent compares the full configuration of the volume (name, driverOpts, and labels) to verify that the volumes are identical. When it is false, the container agent uses Docker’s default behavior, which verifies the volume name only. If a volume is shared across container instances, this should be set to false. For more information, see Docker Volumes (p. 231).

**ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM**

Example values: `ec2_instance`

Default value on Linux: none

Default value on Windows: none

If `ec2_instance` is specified, existing tags defined on the container instance are registered to Amazon ECS. The tags are discoverable using the `ListTagsForResource` operation. The IAM role associated with the container instance should have the `ec2:DescribeTags` action allowed. For more information, see Adding Tags to a Container Instance (p. 362).

**ECS_CONTAINER_INSTANCE_TAGS**

Example values: `{"tag_key": "tag_val"}`

Default value on Linux: {}

Default value on Windows: {}

Metadata applied to container instances to help you categorize and organize your resources. Each tag consists of a custom-defined key and an optional value. Tag keys can have a maximum character length of 128 characters. Tag values can have a maximum length of 256 characters.

If container instance tags are propagated using the `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` parameter, those tags are overwritten by the tags specified using `ECS_CONTAINER_INSTANCE_TAGS`. For more information, see Adding Tags to a Container Instance (p. 362).

**ECS_ENABLE_UNTRACKED_IMAGE_CLEANUP**

Example values: `true` | `false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to allow the ECS agent to delete containers and images that are not part of Amazon ECS tasks.

**ECS_EXCLUDE_UNTRACKED_IMAGE**

Example values: `{"alpine": "latest"}`

Default value on Linux: `{}`

Default value on Windows: `{}`

Comma separated list of images (imageName:tag) that should not be deleted by the ECS agent if `ECS_ENABLE_UNTRACKED_IMAGE_CLEANUP` is true.

**ECS_DISABLE_DOCKER_HEALTH_CHECK**

Example values: `true` | `false`

Default value on Linux: `false`
Default value on Windows: false

Whether to disable the Docker container health check for the ECS Agent.

ECS_NVIDIA_RUNTIME

Example values: nvidia

Default value on Linux: nvidia

Default value on Windows: n/a

The runtime to be used to pass NVIDIA GPU devices to containers. This parameter should not be specified as an environment variable in a task definition if the GPU resource requirements are already specified. For more information, see Working with GPUs on Amazon ECS (p. 38).

Storing Container Instance Configuration in Amazon S3

Amazon ECS container agent configuration is controlled with the environment variables described above. Linux variants of the Amazon ECS-optimized AMI look for these variables in /etc/ecs/ecs.config when the container agent starts and configures the agent accordingly. Certain innocuous 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 they would 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, and 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 allow Amazon S3 read-only access for your container instance role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles and select the IAM role to use for your container instances (this role is likely titled ecsInstanceRole). For more information, see Amazon ECS Container Instance IAM Role (p. 415).
4. To narrow the policy results, on the Attach Policy page, for Filter, type S3.
5. Select the box to the left of the AmazonS3ReadOnlyAccess policy and choose Attach Policy.

To store an ecs.config file in Amazon S3

1. Create an ecs.config file with valid environment variables and values from Amazon ECS Container Agent Configuration (p. 152) using the following format. This example configures private registry authentication. For more information, see Private Registry Authentication for Tasks (p. 251).

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA="{"https://index.docker.io/v1/": {
  "auth":"zg2lM2EXAMPLE76T25DK0t","email":"email@example.com"}}}
```

2. To store your configuration file, create a private bucket in Amazon S3. For more information, see Create a Bucket in the Amazon Simple Storage Service Getting Started Guide.
3. Upload the `ecs.config` file to your S3 bucket. For more information, see Add an Object to a Bucket in the Amazon Simple Storage Service Getting Started 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 by following the steps in Launching an Amazon ECS Container Instance (p. 101). In Step 6.g (p. 103), use the following example script that installs the AWS CLI and copies your configuration file to `/etc/ecs/ecs.config`.

```bash
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

Private Registry Authentication for Container Instances

The Amazon ECS container agent can authenticate with private registries, including Docker Hub, 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 Private Registry Authentication for Tasks (p. 251).

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 Container Instance Configuration in Amazon S3 (p. 163).

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",
      "email": "email@example.com"
    }
  }
}
```

**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` 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":"fQ172MzEXAMPLE7o6T25DU0j","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:

```
API Version 2014-11-13
165
```
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.

```bash
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:

```bash
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={
  "https://index.docker.io/v1/": {
    "username": "my_name",
    "password": "my_password",
    "email": "email@example.com"
  },
  "repo.example-01.com": {
    "username": "my_name",
    "password": "my_password",
    "email": "email@example-01.com"
  },
  "repo.example-02.com": {
    "username": "another_name",
    "password": "another_password",
    "email": "email@example-02.com"
  }
}
```

## Enabling Private Registries

Use the following procedure to enable 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:

   ```bash
   sudo vi /etc/ecs/ecs.config
   ```

   This example authenticates a Docker Hub user account:

   ```bash
   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:

   ```bash
   docker inspect ecs-agent | grep ECS_DATADIR
   ```

   Output:

   ```bash
   "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
Automated 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 (p. 152).

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 (p. 146).

Tunable Parameters

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 (p. 152).

ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION
This variable specifies the time to wait before removing any containers that belong to stopped tasks. The image cleanup process cannot delete an image as long as there is a container that references it. After images are not referenced by any containers (either stopped or running), then the image becomes a candidate for cleanup. By default, this parameter is set to 3 hours but you can reduce this period to as low as 1 minute, if you need to for your application.

ECS_DISABLE_IMAGE_CLEANUP
If you set this variable to true, then automated image cleanup is disabled 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.

Cleanup Workflow

When the Amazon ECS container agent is running and automated image cleanup is not disabled, 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.

Amazon ECS Container Metadata File

Beginning with version 1.15.0 of the Amazon ECS container agent, various container metadata is available within ECS task containers. By enabling this feature, you can query the information about a task, container, and container instance from within the container. You can also get it from the container instance by reading the metadata file for each container. The metadata file is created on the host instance and mounted in the container as a Docker volume.
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:**
  
  ```bash
  cat $ECS_CONTAINER_METADATA_FILE
  ```

- **For Windows instances (PowerShell):**
  
  ```powershell
  Get-Content -path $env:ECS_CONTAINER_METADATA_FILE
  ```

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 Automated Task and Image Cleanup (p. 167).

### Enabling Container Metadata

This feature is disabled by default. You can enable 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 (p. 152).

**Note**

The minimum Amazon ECS container agent version to support this feature is 1.15.0.

If the `ECS_ENABLE_CONTAINER_METADATA` is set to `true` when the agent starts, metadata files are created for any future containers started by ECS.

**Note**

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.

### 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/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's state file is located. For more information, see Amazon ECS Container Agent Configuration (p. 152).

- 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
  ```

### 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.

**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.

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.

Example Amazon ECS container metadata file (READY)

The following example shows a container metadata file in the READY status.

```json
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/2b88376d-aba3-4950-9ddf-bcb0f388a40c",
    "ContainerID": "98e444408169587b826b4cd76c6732e5899747e753af1e19a35db6b4f9e9c32",
    "ContainerName": "metadata",
    "DockerContainerName": "ecs-metadata-7-metadata-f0edbf6d09fdef20800",
    "ImageID": "sha256:c24f66af34b4d765587f7743109e2476b6325f6c6c167c6e1e07cd121a22b341",
    "ImageName": "$httpd:2.4$",
    "PortMappings": [
        {
            "ContainerPort": 80,
            "HostPort": 80,
            "BindIp": 
        },
        {"Protocol": "tcp"
    }
},
"Networks": [
    {"NetworkMode": "bridge",
    "IPv4Addresses": ["172.17.0.2"
    ]
}
```
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",
    "ContainerName": "metadata"
}
```

Amazon ECS Task Metadata Endpoint

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 3** – Available for all tasks that use the EC2 launch type and are launched on Amazon EC2 infrastructure running at least version 1.21.0 of the Amazon ECS container agent. For more information, see Task Metadata Endpoint version 3 (p. 172).
- **Task metadata endpoint version 2** – Available for tasks that use the Fargate launch type on platform version v1.1.0 or later and tasks that use the EC2 launch type that also use the `awsvpc` network mode and are launched on Amazon EC2 infrastructure running at least version 1.17.0 of the Amazon ECS container agent. For more information, see Task Metadata Endpoint version 2 (p. 176).

For information about a sample Go application that queries the metadata and stats API endpoints, see https://github.com/aws/amazon-ecs-agent/blob/2bf4348a0ff89e23be4e82a6c5ff28edf777092c/misc/taskmetadata-validator/taskmetadata-validator.go.

**Topics**

- Task Metadata Endpoint version 3 (p. 172)
- Task Metadata Endpoint version 2 (p. 176)

**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.

**Enabling Task Metadata**

The task metadata version 3 feature is enabled by default for tasks using the EC2 launch type and are launched on Amazon EC2 infrastructure running at least version 1.21.0 of the Amazon ECS container agent. For more information, see Amazon ECS Container Agent Versions (p. 142).
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 (p. 146).

**Important**
For tasks using the Fargate launch type, the task metadata version 2 endpoint is supported. For more information, see Task Metadata Endpoint version 2 (p. 176).

### 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 Task Metadata JSON Response (p. 173).

- **${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.

### Task Metadata JSON Response

The following information is returned from the task metadata endpoint (${ECS_CONTAINER_METADATA_URI}/task) JSON response.

- **Cluster**
  - The Amazon ECS cluster to which the task belongs.

- **TaskARN**
  - The full 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.

- **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 for the image.

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 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.
Limits

The resource limits specified at the task level (such as CPU and memory). This parameter is omitted if no resource limits are defined.

PullStartedAt

The time stamp for when the first container image pull began.

PullStoppedAt

The time stamp for when the last container image pull finished.

ExecutionStoppedAt

The time stamp for when the tasks DesiredStatus moved to STOPPED. This occurs when an essential container moves to STOPPED.

AvailabilityZone

The Availability Zone the task is in.

Example Task Metadata Response

The following JSON response is for a single-container task.

```json
{
    "Cluster": "default",
    "Family": "nginx",
    "Revision": "5",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Containers": [
      {
        "DockerId": "731a0d6a3b4210e2448339bc7015aaa7bfe4fa256384f4102db86ef94cbcc4c",
        "Name": "~internal~ecs~pause",
        "DockerName": "ecs-nginx-5-internalecspause-acc699c0bf2d6d1700",
        "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-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": [
              "10.0.1.106"
            ]
          }
        ]
      }
    ]
}
Task Metadata Endpoint version 2

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 AWS Fargate Platform Versions (p. 36).
- Tasks using the EC2 launch type that also use the `awsvpc` network mode and are launched on Amazon EC2 infrastructure running at least version 1.17.0 of the Amazon ECS container agent. For more information, see Amazon ECS Container Agent Versions (p. 142).
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 (p. 146).

**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 (p. 177).

169.254.170.2/v2/metadata/<container-id>

This endpoint returns metadata JSON for the specified Docker container ID.

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 ECS cluster to which the task belongs.

- **TaskARN**
  The full 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.

- **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 for the image.

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 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.

Limits

The resource limits specified at the task level (such as CPU and memory). This parameter is omitted if no resource limits are defined.
PullStartedAt

The time stamp for when the first container image pull began.

PullStoppedAt

The time stamp for when the last container image pull finished.

ExecutionStoppedAt

The time stamp for when the tasks DesiredStatus moved to STOPPED. This occurs when an essential container moves to STOPPED.

AvailabilityZone

The Availability Zone the task is in.

Example Task Metadata Response

The following JSON response is for a single-container task.

```
{
    "Cluster": "default",
    "Family": "nginx",
    "Revision": "5",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Containers": [
        {
            "DockerId": "731a0d6a3b4210e2448339bc7015aa79bfe4fa256384f4102db86ef94cbcc4c",
            "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-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": "43481a6ce4842ee8fe72fc28500c5b52edcc0917f105b83379f88cac1ff3946",
            "Name": "nginx-curl",
```

API Version 2014-11-13

179
Amazon ECS Container Agent 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 (p. 415).

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.

```
[ec2-user ~]# curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Output:

```json
{
  "Cluster": "default",
  "ContainerInstanceArn": "<container_instance_ARN>",
  "Version": "Amazon ECS Agent - v1.28.0 (d2456beb)"
}
```
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:

```
[ec2-user ~] $ curl http://localhost:51678/v1/tasks
```

Output:

```
{
  "Tasks": [
    {
      "Arn": "arn:aws:ecs:us-east-1:<aws_account_id>:task/example5-58ff-46c9-ae05-543f8example",
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Family": "hello_world",
      "Version": "8",
      "Containers": [
        {
          "DockerId": "9581a69a761a557fbf6ce3d0f6745e6af5b9dbfe8666b2c54af156f1a53f8f11",
          "DockerName": "ecs-hello_world-8-mysql-fc8a8f9f10d8f9d8301",
          "Name": "mysql"
        },
        {
          "DockerId": "bf25c5c5b2d4db48846c7236e75b6915e1e7f88d3161cej6a06831e39814a15",
          "DockerName": "ecs-hello_world-8-wordpress-8f8d098b488d8f9d8301",
          "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.

```
[ec2-user ~] $ curl http://localhost:51678/v1/tasks?dockerid=79c796ed2a7f
```

Output:

```
{
  "Arn": "arn:aws:ecs:us-east-1:<aws_account_id>:task/e01d58a8-151b-40e8-bc01-22647b96ecfec",
  "Containers": [
    {
      "DockerId": "79c796ed2a7f864f885c76f83f3165488097279d296ad05bd5201a1c65b2920",
      "DockerName": "ecs-hello_world-8-wordpress-e8b8d09b488d8f9d8301",
      "Name": "wordpress"
    }
  ]
}
```
HTTP Proxy Configuration

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. The process differs for Linux and Windows instances, so be sure to read the appropriate section below for your application.

Topics

- Amazon Linux Container Instance Configuration (p. 182)
- Windows Container Instance Configuration (p. 185)

Amazon Linux Container Instance Configuration

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 could also manually edit the configuration file and restart the agent afterwards.

/etc/ecs/ecs.config (Amazon Linux 2 and Amazon Linux 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 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=169.254.169.254,169.254.169.254

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"

Set this value to 169.254.169.254 to filter EC2 instance metadata from the proxy.

/etc/sysconfig/docker (Amazon Linux AMI only)
export 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 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

Set this value to 169.254.169.254 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.

Example Amazon Linux HTTP proxy user data script

The example user data cloud-boothook script below configures the Amazon ECS container agent, ecs-init, the Docker daemon, and yum 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 Launching an Amazon ECS Container Instance (p. 101), and in Step 6.g (p. 103). Then, copy and paste the cloud-boothook script below into the User data field (be sure to substitute the red example values with your own proxy and cluster information).

Note
The user data script below only supports Amazon Linux 2 and Amazon Linux AMI variants of the Amazon ECS-optimized AMI.

```bash
#cloud-boothook
# Configure Yum, the Docker daemon, and the ECS agent to use an HTTP proxy

# Specify proxy host, port number, and ECS cluster name to use
PROXY_HOST=10.0.0.131
PROXY_PORT=3128
CLUSTER_NAME=proxy-test

if grep -q 'Amazon Linux release 2' /etc/system-release; then
  OS=AL2
  echo "Setting OS to Amazon Linux 2"
else
  if grep -q 'Amazon Linux AMI' /etc/system-release; then
    OS=ALAMI
    echo "Setting OS to Amazon Linux AMI"
  else
    echo "This user data script only supports Amazon Linux 2 and Amazon Linux AMI."
```
# Set Yum HTTP proxy
if [ ! -f /var/lib/cloud/instance/sem/config_yum_http_proxy ]; then
    echo "proxy=http://$PROXY_HOST:$PROXY_PORT" >> /etc/yum.conf
    echo "## $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_yum_http_proxy
fi

# Set Docker HTTP proxy (different methods for Amazon Linux 2 and Amazon Linux AMI)
# Amazon Linux 2
if [ $OS == "AL2" ] && [ ! -f /var/lib/cloud/instance/sem/config_docker_http_proxy ]; then
    mkdir /etc/systemd/system/docker.service.d
    cat <<EOF > /etc/systemd/system/docker.service.d/http-proxy.conf
    [Service]
    Environment="HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/"
    Environment="HTTPS_PROXY=https://$PROXY_HOST:$PROXY_PORT/"
    Environment="NO_PROXY=169.254.169.254"
    EOF
    systemctl daemon-reload
    if [ "$(systemctl is-active docker)" == "active" ]
    then
        systemctl restart docker
    fi
    echo "## $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_docker_http_proxy
fi

# Amazon Linux AMI
if [ $OS == "ALAMI" ] && [ ! -f /var/lib/cloud/instance/sem/config_docker_http_proxy ]; then
    echo "export HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/
    echo "export HTTPS_PROXY=https://$PROXY_HOST:$PROXY_PORT/
    echo "export NO_PROXY=169.254.169.254" >> /etc/sysconfig/docker
    echo "## $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_docker_http_proxy
fi

# Set ECS agent HTTP proxy
if [ ! -f /var/lib/cloud/instance/sem/config_ecs-agent_http_proxy ]; then
    cat <<EOF > /etc/ecs/ecs.config
    ECS_CLUSTER=$CLUSTER_NAME
    HTTP_PROXY=$PROXY_HOST:$PROXY_PORT
    NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock
    EOF
    echo "## $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_ecs-agent_http_proxy
fi

# Set ecs-init HTTP proxy (different methods for Amazon Linux 2 and Amazon Linux AMI)
# Amazon Linux 2
if [ $OS == "AL2" ] && [ ! -f /var/lib/cloud/instance/sem/config_ecs-init_http_proxy ]; then
    mkdir /etc/systemd/system/ecs.service.d
    cat <<EOF > /etc/systemd/system/ecs.service.d/http-proxy.conf
    [Service]
    Environment="HTTP_PROXY=$PROXY_HOST:$PROXY_PORT/"
    Environment="NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock"
    EOF
    systemctl daemon-reload
    if [ "$(systemctl is-active ecs)" == "active" ]
    then
        systemctl restart ecs
    fi
    echo "## $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_ecs-init_http_proxy
fi

# Amazon Linux AMI
Windows Container Instance Configuration

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).

```
```

Set HTTP_PROXY to the hostname (or IP address) and port number of an HTTP proxy to use for the 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 Step 2: Launching a Windows Container Instance into your Cluster (p. 612). When you reach Step 9 (p. 613), 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 Windows IAM Roles for Tasks (p. 619).

```
<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>
```
Amazon ECS Task Definitions

A task definition is required to run Docker containers in Amazon ECS. Some of the parameters you can specify in a task definition include:

- 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 launch type to use, which determines the infrastructure on which your tasks are hosted
- The Docker networking mode to use for the containers in your task
- The logging configuration to use for your tasks
- Whether the task should continue to run if the container finishes or fails
- The command the container should run when it is started
- Any data volumes that should be used with the containers in the task
- The IAM role that your tasks should use

You can define multiple containers in a task definition. The parameters that you use depend on the launch type you choose for the task. Not all parameters are valid. For more information about the parameters available and which launch types they are valid for in a task definition, see Task Definition Parameters (p. 195).

Your entire application stack does not need to exist on a single task definition, and in most cases it should not. Your application can span multiple task definitions by combining related containers into their own task definitions, each representing a single component. For more information, see Application Architecture (p. 186).

Topics
- Application Architecture (p. 186)
- Creating a Task Definition (p. 188)
- Task Definition Parameters (p. 195)
- Amazon ECS Launch Types (p. 227)
- Using Data Volumes in Tasks (p. 229)
- Task Networking with the awsvpc Network Mode (p. 242)
- Using the awslogs Log Driver (p. 245)
- Private Registry Authentication for Tasks (p. 251)
- Specifying Sensitive Data (p. 254)
- Example Task Definitions (p. 260)
- Updating a Task Definition (p. 266)
- Deregistering Task Definitions (p. 266)

Application Architecture

How you architect your application on Amazon ECS depends on several factors, with the launch type you are using being a key differentiator. We give the following guidance, broken down by launch type, which should assist in the process.
Using the Fargate Launch Type

When architecting your application using the Fargate launch type for your tasks, the main question is when should you put multiple containers into the same task definition versus deploying containers separately in multiple task definitions.

You should put multiple containers in the same task definition if:

• Containers share a common lifecycle (that is, they should be launched and terminated together).
• Containers are required to be run on the same underlying host (that is, one container references the other on a localhost port).
• You want your containers to share resources.
• Your containers share data volumes.

Otherwise, you should define your containers in separate tasks definitions so that you can scale, provision, and deprovision them separately.

Using the EC2 Launch Type

When you're considering how to model task definitions and services using the EC2 launch type, it helps to think about what processes need to run together and how to scale each component.

As an example, imagine an application that 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

In your development environment, you probably run all three containers together on your Docker host. You might be tempted to use the same approach for your production environment, but this approach has several drawbacks:

• Changes to one component can impact all three components, which may be a larger scope for the change than anticipated.
• Each component is more difficult to scale because you have to scale every container proportionally.
• Task definitions can only have 10 container definitions and your application stack might require more, either now or in the future.
• Every container in a task definition must land on the same container instance, which may limit your instance choices to the largest sizes.

Instead, you should create task definitions that group the containers that are used for a common purpose, and separate the different components into multiple task definitions. In this example, three task definitions each specify one container. The example cluster below has three container instances registered with 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, you could add a log streaming container to your front-end service and include that 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 a Service (p. 340). In your services, you can associate containers with Elastic Load Balancing load balancers. For more information, see Service Load Balancing (p. 306). When your application requirements change, you can update your services to scale the number of desired tasks up or down, or to deploy newer versions of the containers in your tasks. For more information, see Updating a Service (p. 350).

Creating a Task Definition

Before you can run Docker containers on Amazon ECS, you must create a task definition. You can define multiple containers and data volumes in a task definition. For more information about the parameters available in a task definition, see Task Definition Parameters (p. 195).

To create a new task definition

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Task Definitions, Create new Task Definition.
3. On the Select compatibilities page, select the launch type that your task should use and choose Next step.
   
   Note
   
   The Fargate launch type is not compatible with Windows containers.

4. Follow the steps under one of the following tabs, according to the launch type that you have chosen.
Fargate launch type

Using the Fargate launch type compatibility template

If you chose Fargate, complete the following steps:

1. (Optional) If you have a JSON representation of your task definition, complete the following steps:
   a. On the Configure task and container definitions page, scroll to the bottom of the page and choose Configure via JSON.
   b. Paste your task definition JSON into the text area and choose Save.
   c. Verify your information and choose Create.

2. For Task Definition Name, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
3. (Optional) For Task Role, choose an IAM role that provides permissions for containers in your task to make calls to AWS API operations on your behalf. For more information, see IAM Roles for Tasks (p. 435).

   Note
   Only roles that have the Amazon EC2 Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 437).
4. For Task execution IAM role, either select your task execution role or choose Create new role so that the console can create one for you. For more information, see Amazon ECS Task Execution IAM Role (p. 418).
5. For Task size, choose a value for Task memory (GB) and Task CPU (vCPU). The table below shows the valid combinations.

<table>
<thead>
<tr>
<th>CPU value</th>
<th>Memory value</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 (.25 vCPU)</td>
<td>512 MB, 1 GB, 2 GB</td>
</tr>
<tr>
<td>512 (.5 vCPU)</td>
<td>1 GB, 2 GB, 3 GB, 4 GB</td>
</tr>
<tr>
<td>1024 (1 vCPU)</td>
<td>2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB</td>
</tr>
<tr>
<td>2048 (2 vCPU)</td>
<td>Between 4 GB and 16 GB in 1 GB increments</td>
</tr>
<tr>
<td>4096 (4 vCPU)</td>
<td>Between 8 GB and 30 GB in 1 GB increments</td>
</tr>
</tbody>
</table>

6. For each container in your task definition, complete the following steps:
   a. Choose Add container.
   b. Fill out each required field and any optional fields to use in your container definitions. More container definition parameters are available in the Advanced container configuration menu. For more information, see Task Definition Parameters (p. 195).
   c. Choose Add to add your container to the task definition.
7. (Optional) To define data volumes for your task, choose Add volume. For more information, see Using Data Volumes in Tasks (p. 229).
   a. For Name, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
8. In the Tags section, specify the key and value for each tag to associate with the task definition. For more information, see Tagging Your Amazon ECS Resources.

9. Choose Create.

EC2 launch type

Using the EC2 launch type compatibility template

If you chose EC2, complete the following steps:

1. (Optional) If you have a JSON representation of your task definition, complete the following steps:
   a. On the Configure task and container definitions page, scroll to the bottom of the page and choose Configure via JSON.
   b. Paste your task definition JSON into the text area and choose Save.
   c. Verify your information and choose Create.

Scroll to the bottom of the page and choose Configure via JSON.

2. For Task Definition Name, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

3. (Optional) For Task Role, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 435).

For tasks that use the EC2 launch type, these permissions are usually granted by the Amazon ECS Container Instance IAM role. For more information, see Amazon ECS Container Instance IAM Role (p. 415).

   Note
Only roles that have the Amazon EC2 Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 437).

4. (Optional) For Network Mode, choose the Docker network mode to use for the containers in your task. The available network modes correspond to those described in Network settings in the Docker run reference.

The default Docker network mode is bridge. If the network mode is set to none, you can't specify port mappings in your container definitions, and the task's containers do not have external connectivity. If the network mode is awsvpc, the task is allocated an elastic network interface. The host and awsvpc network modes offer the highest networking performance for containers because they use the Amazon EC2 network stack instead of the virtualized network stack provided by the bridge mode; however, exposed container ports are mapped directly to the corresponding host port, so you cannot take advantage of dynamic host port mappings or run multiple instantiations of the same task on a single container instance if port mappings are used.

5. (Optional) For Task execution role, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf.

For tasks that use the EC2 launch type, these permissions are usually granted by the Amazon ECS Container Instance IAM role, which is specified earlier as the Task Role. There is no need to specify a task execution role. For more information, see Amazon ECS Task Execution IAM Role (p. 418).
6. (Optional) For **Task size**, choose a value for **Task memory (GB)** and **Task CPU (vCPU)**. Supported **Task CPU (vCPU)** values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

   **Note**
   Task-level CPU and memory parameters are ignored for Windows containers. We recommend specifying container-level resources for Windows containers.

7. For each container in your task definition, complete the following steps.
   
a. Choose **Add container**.
   
b. Fill out each required field and any optional fields to use in your container definitions (more container definition parameters are available in the **Advanced container configuration** menu). For more information, see **Task Definition Parameters** (p. 195).
   
c. Choose **Add** to add your container to the task definition.

8. (Optional) For **Constraint**, define how tasks that are created from this task definition are placed in your cluster. For tasks that use the EC2 launch type, you can use constraints to place tasks based on Availability Zone, instance type, or custom attributes. For more information, see **Amazon ECS Task Placement Constraints** (p. 276).

9. (Optional) To define data volumes for your task, choose **Add volume**. You can create either a bind mount or Docker volume. For more information, see **Using Data Volumes in Tasks** (p. 229).
   
a. For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
   
b. (Optional) To create a bind mount volume, for **Source path**, type the path on the host container instance to present to the container. If you leave this field empty, the Docker daemon assigns a host path for you. If you specify a source path, the data volume persists at the specified location on the host container instance until you delete it manually. If the source path does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported to the container.
   
c. To create a Docker volume, select **Specify a volume driver**.
      
      i. For **Driver**, choose the Docker volume driver to use. The driver value must match the driver name provided by Docker. Use docker plugin ls on your container instance to retrieve the driver name.
      
      ii. For **Scope**, choose the option that determines the lifecycle of the Docker volume. 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.
      
      iii. Select **Enable auto-provisioning** to have the Docker volume created if it does not already exist. This option is only available for volumes that specify the shared scope.
      
      iv. For **Driver options**, specify the driver-specific key values to use.
      
      v. For **Volume labels**, specify the custom metadata to add to your Docker volume.

10. In the **Tags** section, specify the key and value for each tag to associate with the task definition. For more information, see **Tagging Your Amazon ECS Resources**.

11. Choose **Create**.

### Task Definition Template

An empty task definition template is shown below. 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 **Task Definition Parameters** (p. 195).

---

API Version 2014-11-13
191
{
    "family": "",
    "taskRoleArn": "",
    "executionRoleArn": "",
    "networkMode": "none",
    "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": ""
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
                }
            ],
            "volumesFrom": [
                {
                    "sourceContainer": "",
                    "readOnly": true
                }
            ],
            "linuxParameters": {
                "capabilities": {
                    "add": [],
                    "drop": []
                },
                "devices": [
                    {
                        "hostPath": "",
                        "containerPath": "",
                        "permissions": []
                    }
                ]
            }
        }
    ]
}

API Version 2014-11-13
192
"mknod"
]
},
"initProcessEnabled": true,
"sharedMemorySize": 0,
"tmpfs": [
  {
    "containerPath": ",",
    "size": 0,
    "mountOptions": [
      ""
    ]
  }
],
"secrets": [
  {
    "name": ",",
    "valueFrom": ","
  }
],
"dependsOn": [
  {
    "containerName": ",",
    "condition": "SUCCESS"
  }
],
"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": "rss",
    "softLimit": 0,
    "hardLimit": 0
  }
],
"logConfiguration": {
  "logDriver": "syslog",
  "options": {"API Version 2014-11-13"}}
"KeyName": "",
{
  "healthCheck": {
    "command": [
      ""
    ],
    "interval": 0,
    "timeout": 0,
    "retries": 0,
    "startPeriod": 0
  },
  "systemControls": [
    {
      "namespace": "",
      "value": ""
    }
  ],
  "resourceRequirements": [
    {
      "value": "",
      "type": "GPU"
    }
  ]
},
"volumes": [
  {
    "name": "",
    "host": {
      "sourcePath": ""
    },
    "dockerVolumeConfiguration": {
      "scope": "task",
      "autopropagation": true,
      "driver": "",
      "driverOptions": {
        "KeyName": ""
      },
      "labels": {
        "KeyName": ""
      }
    }
  }
],
"placementConstraints": [
  {
    "type": "memberOf",
    "expression": ""
  }
],
"requiresCompatibilities": [
  "FARGATE"
],
"cpu": "",
"memory": "",
"tags": [
  {
    "key": "",
    "value": ""
  }
],
"pidMode": "host",
"ipcMode": "host",
"proxyConfiguration": {
  "type": "APPmesh"
Task Definition Parameters

Task definitions are split into separate parts: the task family, the IAM task role, the network mode, container definitions, volumes, task placement constraints, and launch types. The family is the name of the task, and each family can have multiple revisions. The IAM task role specifies the permissions that containers in the task should have. The network mode determines how the networking is configured for your containers. Container definitions specify which image to use, how much CPU and memory the container are allocated, and many more options. Volumes allow you to share data between containers and even persist the data on the container instance when the containers are no longer running. The task placement constraints customize how your tasks are placed within the infrastructure. The launch type determines which infrastructure your tasks use.

The family and container definitions are required in a task definition, while task role, network mode, volumes, task placement constraints, and launch type are optional.

Parts
- Family (p. 195)
- Task Role (p. 196)
- Task Execution Role (p. 196)
- Network Mode (p. 196)
- Container Definitions (p. 197)
- Volumes (p. 220)
- Task Placement Constraints (p. 222)
- Launch Types (p. 222)
- Task Size (p. 223)
- Other Task Definition Parameters (p. 224)

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 is
registered into a particular family is given a revision of 1, and any task definitions registered after that are given a sequential revision number.

**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 IAM Roles for Tasks (p. 435).

IAM roles for tasks on Windows require that the `-EnableTaskIAMRole` option is set when you launch the Amazon ECS-optimized Windows AMI. Your containers must also run some configuration code in order to take advantage of the feature. For more information, see Windows IAM Roles for Tasks (p. 619).

**Task Execution Role**

`executionRoleArn`

Type: string

Required: no

When you register a task definition, you can provide a task execution role that allows the containers in the task to pull container images and publish container logs to CloudWatch on your behalf. For more information, see Amazon ECS Task Execution IAM Role (p. 418).

**Network Mode**

`networkMode`

Type: string

Required: no

The Docker networking mode to use for the containers in the task. The valid values are `none`, `bridge`, `awsvpc`, and `host`. The default Docker network mode is `bridge`.

If the network mode is set to `none`, the task's containers do not have external connectivity and port mappings can't be specified in the container definition.

If the network mode is `bridge`, the task utilizes Docker's built-in virtual network which runs inside each container instance.

If the network mode is `host`, the task bypasses Docker's built-in virtual network and maps container ports directly to the EC2 instance's network interface directly. In this mode, you can't run multiple instantiations of the same task on a single container instance when port mappings are used.

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 Task Networking with the awsvpc Network Mode (p. 242). Currently, only the Amazon ECS-optimized AMI, other Amazon Linux variants with the ecs-init package, or AWS Fargate infrastructure support the awsvpc network mode.

The host and awsvpc network modes offer the highest networking performance for containers because they use the Amazon EC2 network stack instead of the virtualized network stack provided by the bridge mode. 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), so you cannot take advantage of dynamic host port mappings.

Docker for Windows uses a different network mode (known as NAT) than Docker for Linux. When you register a task definition with Windows containers, you must not specify a network mode. If you use the AWS Management Console to register a task definition with Windows containers, you must choose the default network mode.

If using the Fargate launch type, the awsvpc network mode is required. If using the EC2 launch type, the allowable network mode depends on the underlying EC2 instance's operating system. If Linux, any network mode can be used. If Windows, only the NAT mode is allowed, as described above.

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 (p. 197)
- Advanced Container Definition Parameters (p. 201)
- Other Container Definition Parameters (p. 213)

Standard Container Definition Parameters

The following task definition parameters are either required or used in most container definitions.

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 are linking multiple containers together in a task definition, the name of one container can be entered in the links of another container to connect the containers.

image

Type: string
Required: yes

The image used to start a container. This string is passed directly to the Docker daemon. Images in the Docker Hub registry are available by default. 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 Create a container section of the Docker Remote API and the IMAGE parameter of docker run.
• 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 are not propagated to already running tasks.

• Images in private registries are supported. For more information, see Private Registry Authentication for Tasks (p. 251).

• 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@sha256:94afd1f2e64d908bc90dcbca0035a5b567EXAMPLE

• 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).

memory

Type: integer

Required: no

The hard limit (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. This parameter maps to Memory in the Create a container section of the Docker Remote API and the --memory option to docker run.

If your containers are part of a task using the Fargate launch type, this field is optional and the only requirement is that the total amount of memory reserved for all containers within a task be lower than the task memory value.

For containers that are part of a task using the EC2 launch type, you must specify a non-zero integer for one or both of memory or memoryReservation in container definitions. 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 on which the container is placed; otherwise, the value of memory is used.

The Docker daemon reserves a minimum of 4 MiB of memory for a container, so you should not specify fewer than 4 MiB of memory for your containers.

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 Container Instance Memory Management (p. 124).

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 consume more memory when needed, up to either the hard limit 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 Create a container section of the Docker Remote API and the --memory-reservation option to docker run.

You must specify a non-zero integer for one or both of memory or memoryReservation in container definitions. 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.
resources for the container instance on which the container is placed; otherwise, the value of memory is used.

For example, if 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 would allow the container to only reserve 128 MiB of memory from the remaining resources on the container instance, but also allow the container to consume more memory resources when needed.

The Docker daemon reserves a minimum of 4 MiB of memory for a container, so you should not specify fewer than 4 MiB of memory for your containers.

**portMappings**

Type: object array

Required: no

Port mappings allow containers to access ports on the host container instance to send or receive traffic.

For task definitions that use the awsvpc network mode, you should only specify the containerPort. The hostPort can be left blank or it must be the same value as the containerPort.

Port mappings on Windows use the NetNAT gateway address rather than localhost. There is no loopback for port mappings on Windows, so you cannot access a container’s mapped port from the host itself.

This parameter maps to PortBindings in the Create a container section of the Docker Remote API and the --publish option to docker run. If the network mode of a task definition is set to host, then host ports must either be undefined or they must 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.

**containerPort**

Type: integer

Required: yes, when portMappings are used

The port number on the container that is bound to the user-specified or automatically assigned host port.

If using containers in a task with the Fargate launch type, exposed ports should be specified using containerPort.

If using containers in a task with the EC2 launch type and you specify a container port and not a host port, 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 do not count toward the 100 reserved ports limit of a container instance.

**hostPort**

Type: integer
Required: no

The port number on the container instance to reserve for your container.

If using containers in a task with the Fargate launch type, the `hostPort` can either be left blank or be the same value as `containerPort`.

If using containers in a task with the EC2 launch type, 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` and your container automatically receives a port (this is referred to as dynamic host port mapping) in the ephemeral port range for your container instance operating system and Docker version.

The default ephemeral port range is 49153–65535, and this range is used for Docker versions before 1.6.0. For Docker version 1.6.0 and later, the Docker daemon tries to read the ephemeral port range from `/proc/sys/net/ipv4/ip_local_port_range` (which is 32768–61000 on the latest Amazon ECS-optimized AMI); if this kernel parameter is unavailable, the default ephemeral port range is used. Do not attempt to specify a host port in the ephemeral port range, as these are reserved for automatic assignment. In general, ports below 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, and a container instance may have up to 100 reserved ports at a time, including the default reserved ports. Automatically assigned ports do not count toward the 100 reserved ports limit.

### protocol

- **Type**: string
- **Required**: no

The protocol used for the port mapping. Valid values are `tcp` and `udp`. The default is `tcp`.

**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 (p. 146).

If you are specifying a host port, use the following syntax:

```json
"portMappings": [
  {
    "containerPort": integer,
    "hostPort": integer
  }
  ...
]
```

If you want an automatically assigned host port, use the following syntax:

```json
"portMappings": [
  {
    "containerPort": integer
  }
  ...
]
```
Advanced Container Definition Parameters

The following advanced container definition parameters provide extended capabilities to the `docker run` command that is used to launch containers on your Amazon ECS container instances.

**Topics**

- Health Check (p. 201)
- Environment (p. 202)
- Network Settings (p. 205)
- Storage and Logging (p. 207)
- Security (p. 211)
- Resource Limits (p. 212)
- Docker Labels (p. 213)

**Health Check**

`healthCheck`

The health check command and associated configuration parameters for the container. This parameter maps to `HealthCheck` in the Create a container section of the Docker Remote API and the `HEALTHCHECK` parameter of `docker run`.

**Note**

The Amazon ECS container agent only monitors and reports on the health checks specified in the task definition. Amazon ECS does not monitor Docker health checks that are embedded in a container image and not 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.

Task health is reported by the `healthStatus` of the task, which is determined by the health of the essential containers in the task. If all essential containers in the task are reporting as `HEALTHY`, then the task status also reports as `HEALTHY`. If any essential containers in the task are reporting as `UNHEALTHY` or `UNKNOWN`, then the task status also reports as `UNHEALTHY` or `UNKNOWN`, accordingly. If a service's task reports as unhealthy, it is removed from a service and replaced.

The following are notes about container health check support:

- Container health checks require version 1.17.0 or greater of the Amazon ECS container agent. For more information, see Updating the Amazon ECS Container Agent (p. 146).
- Container health checks are supported for Fargate tasks if you are using platform version 1.1.0 or later. For more information, see AWS Fargate Platform Versions (p. 36).
- Container health checks are not supported for tasks that are part of a service that is configured to use a Classic Load Balancer.

**command**

A string array representing the command that the container runs to determine if it is healthy. The string array can start with `CMD` to execute the command arguments directly, or `CMD-SHELL` to run the command with the container's default shell. If neither is specified, `CMD` is used by default.

When registering a task definition in the AWS Management Console, use a comma separated list of commands which will automatically converted to a string after the task definition is created. An example input for a health check could be:
When registering a task definition using the AWS Management Console JSON panel, the AWS CLI, or the APIs, you should enclose the list of commands in brackets. An example input for a health check could be:

```
[ "CMD-SHELL", "curl -f http://localhost/ || exit 1" ]
```

An exit code of 0 indicates success, and a non-zero exit code indicates failure. For more information, see HealthCheck in the Create a container section of the Docker Remote API.

### interval

The time period in seconds between each health check execution. You may specify between 5 and 300 seconds. The default value is 30 seconds.

### timeout

The time period in seconds to wait for a health check to succeed before it is considered a failure. You may 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 may specify between 1 and 10 retries. The default value is three retries.

### startPeriod

The optional grace period within which to provide containers time to bootstrap before failed health checks count towards the maximum number of retries. You may specify between 0 and 300 seconds. The `startPeriod` is disabled by default.

### Environment

#### cpu

Type: integer

Required: no

The number of CPU units the Amazon ECS container agent will reserve for the container. This parameter maps to `CpuShares` in the Create a container section of the Docker Remote API and the `--cpu-shares` option to `docker run`.

This field is optional for tasks using the Fargate launch type, and the only requirement is that the total amount of CPU reserved for all containers within a task be lower than the task-level CPU value.

**Note**

You can determine the number of CPU units that are available per Amazon EC2 instance type by multiplying 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, if you run a single-container task on a single-core instance type with 512 CPU units specified for that container, and that is the only task running on the container instance, that container could use the full 1,024 CPU unit share at any given time. However, if you launched another copy of the same task on that container instance, each task would be guaranteed a minimum of 512 CPU units when needed, and each container could float to higher CPU usage if the other container was not using it, but if both tasks were 100% active all of the time, they would be 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. For more information, see CPU share constraint in the Docker documentation. The minimum valid CPU share value that the Linux kernel allows is 2. However, the CPU parameter is not required, and you can use CPU values below 2 in your container definitions. For CPU values below 2 (including null), 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, which Docker then converts to 1,024 CPU shares. CPU values of 1 are passed to Docker as 1, which the Linux kernel converts to two CPU shares.
- **Agent versions >= 1.2.0**: Null, zero, and CPU values of 1 are passed to Docker as two CPU shares.

On Windows container instances, the CPU limit is enforced as an absolute limit, or a quota. Windows containers only have access to the specified amount of CPU that is described in the task definition.

### gpu

**Type:** ResourceRequirement object

**Required:** no

The number of physical GPUs the Amazon ECS container agent will reserve for the container. The number of GPUs reserved for all containers in a task should not exceed the number of available GPUs on the container instance the task is launched on. For more information, see Working with GPUs on Amazon ECS (p. 38).

**Note**

This parameter is not supported for Windows containers or tasks using the Fargate launch type.

### essential

**Type:** Boolean

**Required:** no

If the essential parameter of a container is marked as true, and that container fails or stops for any reason, 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 does not 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. If you have an application that is composed of multiple containers, you should group containers that are used for a common purpose into components, and separate the different components into multiple task definitions. For more information, see Application Architecture (p. 186).

```
"essential": true|false
```

### entryPoint

**Important**

Early versions of the Amazon ECS container agent do not 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 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. For
more information about the Docker ENTRYPOINT parameter, go to https://docs.docker.com/engine/reference/builder/#entrypoint.

```
"entryPoint": ["string", ...]
```

**command**

Type: string array

Required: no

The command that is passed to the container. This parameter maps to `Cmd` in the Create a container section of the Docker Remote API and the `COMMAND` parameter to `docker run`. For more information about the Docker `CMD` parameter, go to https://docs.docker.com/engine/reference/builder/#cmd. If there are multiple arguments, each argument should be a separated string in the array.

```
"command": ["string", ...]
```

**workingDirectory**

Type: string

Required: no

The working directory in which to run commands inside the container. 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"
```

**environment**

Type: object array

Required: no

The environment variables to pass to a container. This parameter maps to `Env` in the Create a container section of the Docker Remote API and the `--env` option to `docker run`.

**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" : [ ]
```
secrets

Type: Object array

Required: No

An object representing the secret to expose to your container. For more information, see Specifying Sensitive Data (p. 254).

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 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 exists in the same Region as the task you are launching then 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 disabled within the container. This parameter maps to NetworkDisabled in the Create a container section of the Docker Remote API.

Note

This parameter is not supported for Windows containers or tasks using the awsvpc network mode.

"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. 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. For more information about linking Docker containers, go to https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/. This parameter maps to `Links` in the Create a container section of the Docker Remote API and the `--link` option to `docker run`.

**Note**
This parameter is not supported for Windows containers or tasks using the `awsvpc` network mode.

**Important**
Containers that are collocated on a single container instance may be able to communicate with each other without requiring links or host port mappings. Network isolation is achieved on the container instance using 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 Create a container section of the Docker Remote API and the `--hostname` option to `docker run`.

**Note**
The `hostname` parameter is not supported if you are using the `awsvpc` network mode.

```
"hostname": "string"
```

dnsServers

Type: string array
Required: no

A list of DNS servers that are presented to the container. This parameter maps to `Dns` in the Create a container section of the Docker Remote API and the `--dns` option to `docker run`.

**Note**
This parameter is not 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 Create a container section of the Docker Remote API and the --dns-search option to docker run.

**Note**
This parameter is not supported for Windows containers or tasks using 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 Create a container section of the Docker Remote API and the --add-host option to docker run.

**Note**
This parameter is not 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 Create a container section of the Docker Remote API and the --read-only option to docker run.
### Note

This parameter is not supported for Windows containers.

```
"readonlyRootFileSystem": true|false
```

**mountPoints**

- **Type:** Object
- **Required:** No

The mount points for data volumes in your container.

This parameter maps to **Volumes** in the Create a container section of the Docker Remote 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 point cannot be across drives.

**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 on the container to mount the volume at.

**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**

- **Type:** object array
- **Required:** no

Data volumes to mount from another container. This parameter maps to **VolumesFrom** in the Create a container section of the Docker Remote API 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 tasks using the Fargate launch type, the supported log drivers are awslogs and splunk.

For tasks using the EC2 launch type, the supported log drivers are awslogs, syslog, gelf, fluentd, splunk, journald, and json-file.

For more information on using the awslogs log driver in task definitions to send your container logs to CloudWatch Logs, see Using the awslogs Log Driver (p. 245).

For example task definitions, see Example Task Definitions (p. 260).

This parameter maps to LogConfig in the Create a container section of the Docker Remote API and the --log-driver option to docker run. By default, containers use the same logging driver that the Docker daemon uses; however the container may 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). For more information on the options for different supported log drivers, see Configure logging drivers in the Docker documentation.

The following should be noted when specifying a log configuration for your containers:

• Amazon ECS currently supports a subset of the logging drivers available to the Docker daemon (shown in the valid values below). Additional log drivers may be available in future releases of the Amazon ECS container agent.

• This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

• For tasks using the EC2 launch type, the Amazon ECS container agent running on a container instance must register the logging drivers available on that instance with the ECS_AVAILABLE_LOGGING_DRIVERS environment variable before containers placed on that instance can use these log configuration options. For more information, see Amazon ECS Container Agent Configuration (p. 152).

• For tasks using the Fargate launch type, because you do not have access to the underlying infrastructure your tasks are hosted on, any additional software needed will have to be installed outside of the task. For example, the Fluentd output aggregators or a remote host running Logstash to send Gelf logs to.

"logConfiguration": {
"logDriver": "json-file" | "syslog" | "journald" | "gelf" | "fluentd" | "awslogs" | "splunk",
"options": {
  "string": "string"
  ...
},
"secretOptions": [
  {
    "name": "string",
    "valueFrom": "string"
  }
]

logDriver
Type: string
Valid values: "json-file" | "syslog" | "journald" | "gelf" | "fluentd" | "awslogs" | "splunk"
Required: yes, when logConfiguration is used
The log driver to use for the container. The valid values listed earlier are log drivers that the Amazon ECS container agent can communicate with by default.
For tasks using the Fargate launch type, the supported log drivers are awslogs and splunk.
For tasks using the EC2 launch type, the supported log drivers are awslogs, syslog, gelf, fluentd, splunk, journald, and json-file.

Note
If you have a custom driver that is not listed, you can fork the Amazon ECS container agent project that is available on GitHub and customize it to work with that driver. We encourage you to submit pull requests for changes that you would like to have included. However, we do not currently provide support for 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 configuration options to send to the log driver.
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 representing the secret to pass to the log configuration. For more information, see Specifying Sensitive Data (p. 254).

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"
    }]
  }

Security

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).

  This parameter maps to Privileged in the Create a container section of the Docker Remote API and the --privileged option to docker run.

  Note
  This parameter is not supported for Windows containers or tasks using the Fargate launch type.

  "privileged": true|false

user

  Type: string
  Required: no

  The user name to use inside the container. This parameter maps to User in the Create a container section of the Docker Remote API and the --user option to docker run.

  You can use 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
Required: no
A list of strings to provide custom labels for SELinux and AppArmor multi-level security systems.

This parameter maps to `SecurityOpt` in the Create a container section of the Docker Remote API and the `--security-opt` option to `docker run`.

Note
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

"dockerSecurityOptions": ["string", ...]

Note
The Amazon ECS container agent running on a container instance must register with the `ECS_SELINUX_CAPABLE=true` or `ECS_APPARMOR_CAPABLE=true` environment variables before containers placed on that instance can use these security options. For more information, see Amazon ECS Container Agent Configuration (p. 152).

Resource Limits

ulimits

Type: object array
Required: no
A list of ulimits to set in the container. This parameter maps to `Ulimits` in the Create a container section of the Docker Remote API 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"|"rtprio"|"rttime"|"sigpending"|"stack",
    "softLimit": integer,
    "hardLimit": integer
  }
  ...
]
name

Type: string

Valid values: "core" | "cpu" | "data" | "fs蚝" | "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.

softLimit

Type: integer

Required: yes, when ulimits are used

The soft limit for the ulimit type.

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 Create a container section of the Docker Remote API 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 are able to be used when registering task definitions in the Amazon ECS console by using the Configure via JSON option. For more information, see Creating a Task Definition (p. 188).

Topics
- Linux Parameters (p. 214)
- Container Dependency (p. 216)
- Container Timeouts (p. 217)
- System Controls (p. 218)
- Interactive (p. 219)
Linux Parameters

linuxParameters

Type: LinuxParameters object

Required: no

Linux-specific options that are applied to the container, such as KernelCapabilities.

**Note**

This parameter is not 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 the default capabilities and the non-default available capabilities, see Runtime privilege and Linux capabilities in the Docker run reference. For more detailed information about these Linux capabilities, see the capabilities(7) Linux manual page.

**Note**

If you are using tasks that use the Fargate launch type, capabilities is supported but the add parameter described below is not supported.

**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 Create a container section of the Docker Remote API and the --cap-add option to docker run.

**Note**

If you are using tasks that use the Fargate launch type, the add parameter is not supported.
**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 provided by Docker. This parameter maps to CapDrop in the Create a container section of the Docker Remote API and the --cap-drop option to docker run.

**devices**

Any host devices to expose to the container. This parameter maps to Devices in the Create a container section of the Docker Remote API and the --device option to docker run.

**Note**

If you are using tasks that use the Fargate launch type, the devices parameter is not supported.

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 at which to expose the host device.

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.
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 are using tasks that use the Fargate launch type, the `sharedMemorySize` parameter is not supported.

Type: Integer

tmpfs

The container path, mount options, and size (in MiB) of the tmpfs mount. This parameter maps to the --tmpfs option to `docker run`.

**Note**
If you are using tasks that use the Fargate launch type, the `tmpfs` parameter is not 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" | "dirsycn" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nosecretatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"

size

The 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 Example: Container Dependency (p. 264).

For tasks using the EC2 launch type, the container 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 (p. 146). If you are 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, then they contain the required versions of the container agent and ecs-init. For more information, see Amazon ECS-optimized AMIs (p. 62).

For tasks using the Fargate launch type, this parameter requires that the task or service uses platform version 1.3.0 or later.

```
"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. It 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 nonessential containers that run a script and then exit.
- **SUCCESS** – This condition is the same as COMPLETE, but it also requires that the container exits with a zero status.
- **HEALTHY** – This condition validates that the dependent container passes its Docker healthcheck before permitting other containers to start. This requires that the dependent container has health checks configured. This condition is confirmed only at task startup.

**Container Timeouts**

**startTimeout**

Type: Integer

Required: no
Time duration 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 does not reach the desired status within that time then containerA will give up and not start. This results in the task transitioning to a STOPPED state.

For tasks using the EC2 launch type, the container instances require at least version 1.26.0 of the container agent to enable a container start timeout value. 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 (p. 146). If you are 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, then they contain the required versions of the container agent and ecs-init. For more information, see Amazon ECS-optimized AMIs (p. 62).

For tasks using the Fargate launch type, this parameter requires that the task or service uses platform version 1.3.0 or later.

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. For tasks using the Fargate launch type, the max stop timeout value is 120 seconds. For tasks using the Fargate launch type, this parameter requires that the task or service uses platform version 1.3.0 or later.

For tasks using the EC2 launch type, the stop timeout value for the container takes precedence over the ECS_CONTAINER_STOP_TIMEOUT container agent configuration parameter, if 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 checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 146). If you are 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, then they contain the required versions of the container agent and ecs-init. For more information, see Amazon ECS-optimized AMIs (p. 62).

System Controls

systemControls

Type: SystemControl object

Required: no

A list of namespaced kernel parameters to set in the container. This parameter maps to Sysctls in the Create a container section of the Docker Remote API and the --sysctl option to docker run.

It is not recommended that you specify network-related systemControls parameters for multiple containers in a single task that also uses either the awsvpc or host network mode for the following reasons:

- For tasks that use the awsvpc network mode, if you set systemControls for any container it will apply to all containers in the task. If you set different systemControls for multiple containers in a single task, the container that is started last will determine which systemControls take effect.
• For tasks that use the host network mode, the network namespace systemControls are not supported.

If you are setting an IPC resource namespace to use for the containers in the task, the following will apply to your system controls. For more information, see IPC Mode (p. 224).
• For tasks that use the host IPC mode, IPC namespace systemControls are not supported.
• For tasks that use the task IPC mode, IPC namespace systemControls values will apply to all containers within a task.

Note
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

```
"systemControls": [
  {
    "namespace": "String",
    "value": "string"
  }
]
```

namespace
Type: String
Required: no
The namespaced kernel parameter to set a value for.

Valid IPC namespace values: "kernel.msgmax" | "kernel.msgmnb" | "kernel.msgmni" | "kernel.sem" | "kernel.shmall" | "kernel.shmmmax" | "kernel.shmmax" | "kernel.shm_rmid_forced", as well as Sysctls beginning with "fs.mqueue.*"

Valid network namespace values: Sysctls beginning with "net.*"

value
Type: String
Required: no
The value for the namespaced kernel parameter specified in namespace.

Interactive

interactive
Type: Boolean
Required: no
When this parameter is true, this allows you to deploy containerized applications that require stdin or a tty to be allocated. This parameter maps to OpenStdin in the Create a container section of the Docker Remote API and the --interactive option to docker run.

Pseudo Terminal

pseudoTerminal
Type: Boolean
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:

- **Docker volumes** — A Docker-managed volume that is created under `/var/lib/docker/volumes` on the container 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 only supported when using the EC2 launch type. Windows containers only support the use of the `local` driver. To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see Using volumes.

- **Bind mounts** — A file or directory on the host machine is mounted into a container. Bind mount host volumes are supported when using either the EC2 or Fargate launch types. To use bind mount host volumes, specify a `host` and optional `sourcePath` value in your task definition. For more information, see Using bind mounts.

For more information, see Using Data Volumes in Tasks (p. 229).

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 container definition `mountPoints`.

**dockerVolumeConfiguration**

Type: Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are only supported when using the EC2 launch type. Windows containers only support 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.
Volumes

**autoprovısіon**

Type: Boolean

Default value: false

Required: No

If this value is true, the Docker volume is created if it does not already exist.

**Note**

This field is only used if the scope is shared.

**driver**

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because it is used for task placement. If the driver was installed using the Docker plugin CLI, use `docker plugin ls` to retrieve the driver name from your container instance. If the driver was installed using another method, use Docker plugin discovery to retrieve the driver name. For more information, see Docker plugin discovery. This parameter maps to `Driver` in the Create a volume section of the Docker Remote API and the `--driver` option to `docker volume create`.

**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 the Docker Remote API and the `--opt` option to `docker volume create`.

**labels**

Type: String

Required: No

Custom metadata to add to your Docker volume. This parameter maps to `Labels` in the Create a volume section of the Docker Remote API and the `--label` option to `docker volume create`.

**host**

Required: No

This parameter is specified when using bind mounts. To use Docker volumes, specify a `dockerVolumeConfiguration` instead. The contents of the `host` parameter determine whether your bind mount data volume persists on the host container instance 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.

Bind mount host volumes are supported when using either the EC2 or Fargate launch types.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives. For example, you can mount `C:\my\path:C:\my\path` and `D:\:D:\`, but not `D:\my\path:C:\my\path or D:\:C:\my\path`.
sourcePath
Type: String
Required: No

When the host parameter is used, specify a sourcePath to declare the path on the host container instance that is presented to the container. If this parameter is empty, then the Docker daemon has assigned 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 container instance until you delete it manually. If the sourcePath value does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

Task Placement Constraints

When you register a task definition, you can provide task placement constraints that customize how Amazon ECS places tasks.

If you are using the Fargate launch type, task placement constraints are not supported. By default Fargate tasks are spread across Availability Zones.

For tasks that use the EC2 launch type, you can use constraints to place tasks based on Availability Zone, instance type, or custom attributes. For more information, see Amazon ECS Task Placement Constraints (p. 276).

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 Cluster Query Language (p. 280).

type
Type: string
Required: yes

The type of constraint. Use memberOf to restrict the selection to a group of valid candidates.

Launch Types

When you register a task definition, you specify the launch type to use for your task. For more information, see Amazon ECS Launch Types (p. 227).

The following parameter is allowed in a task definition:

requiresCompatibilities
Type: string array
Required: no

Valid Values: EC2 | FARGATE
The launch type the task is using. This enables a check to ensure that all of the parameters used in the task definition meet the requirements of the launch type.

Valid values are FARGATE and EC2. For more information about launch types, see Amazon ECS Launch Types (p. 227).

**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. If using the EC2 launch type, these fields are optional. If using the Fargate launch type, these fields are required and there are specific values for both CPU and memory that are supported.

**Note**
Task-level CPU and memory parameters are ignored for Windows containers. We recommend specifying container-level resources for Windows containers.

The following parameter is allowed in a task definition:

**cpu**

Type: string

Required: no

**Note**
This parameter is not supported for Windows containers.

The number of CPU units used by the task. It can be expressed as an integer using CPU units, for example 1024, or as a string using vCPUs, for example 1 vCPU or 1 vcpu, in a task definition. When the task definition is registered, a vCPU value is converted to an integer indicating the CPU units.

If using the EC2 launch type, this field is optional. Supported values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

If using the Fargate launch type, this field is required and you must use one of the following values, which determines your range of supported values for the memory parameter:

<table>
<thead>
<tr>
<th>CPU value</th>
<th>Memory value (MiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 (.25 vCPU)</td>
<td>512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)</td>
</tr>
<tr>
<td>512 (.5 vCPU)</td>
<td>1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)</td>
</tr>
<tr>
<td>1024 (1 vCPU)</td>
<td>2048 (2 GB), 3072 (3 GB), 4096 (4 GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)</td>
</tr>
<tr>
<td>2048 (2 vCPU)</td>
<td>Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)</td>
</tr>
<tr>
<td>4096 (4 vCPU)</td>
<td>Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)</td>
</tr>
</tbody>
</table>

**memory**

Type: string
Other Task Definition Parameters

The following task definition parameters are able to be used when registering task definitions in the Amazon ECS console by using the Configure via JSON option. For more information, see Creating a Task Definition (p. 188).

Topics
- IPC Mode (p. 224)
- PID Mode (p. 225)
- Proxy Configuration (p. 225)

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 containers 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 containers 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. For more information, see IPC settings in the Docker run reference.

If the host IPC mode is used, be aware that there is a heightened risk of undesired IPC namespace exposure. For more information, see Docker security.

If you are setting namespaced kernel parameters using systemControls for the containers in the task, the following will apply to your IPC resource namespace. For more information, see System Controls (p. 218).

- For tasks that use the host IPC mode, IPC namespace related systemControls are not supported.
- For tasks that use the task IPC mode, IPC namespace related systemControls will 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

Required: No

The process namespace to use for the containers in the task. The valid values are host or task. If host is specified, then all containers within the tasks that specified the host PID mode on the same container instance share the same IPC resources 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 more information, see PID settings in the Docker run reference.

If the host PID mode is used, be aware that there is a heightened risk of undesired process namespace exposure. For more information, see Docker security.

**Note**
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

**Proxy Configuration**

**proxyConfiguration**

Type: ProxyConfiguration object

Required: no

The configuration details for the App Mesh proxy.

For tasks using 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 AMIs (p. 62).

For tasks using the Fargate launch type, this feature requires that the task or service uses platform version 1.3.0 or later.
Other Task Definition Parameters

**Note**
This parameter is not supported for Windows containers.

```json
"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 will serve 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 egress traffic going to these specified ports is ignored and not redirected to the **ProxyEgressPort**. It can be an empty list.
    - **EgressIgnoredIPs** – (Required) The egress 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.

Amazon ECS Launch Types

An Amazon ECS launch type determines the type of infrastructure on which your tasks and services are hosted.

Fargate Launch Type

The Fargate launch type allows you to run your containerized applications without the need to provision and manage the backend infrastructure. Just register your task definition and Fargate launches the container for you.

The AWS Fargate launch type is currently available in the following Regions:

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
</tr>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
</tr>
<tr>
<td>EU (Ireland)</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>EU (London)</td>
<td>eu-west-2</td>
</tr>
<tr>
<td>EU (Frankfurt)</td>
<td>eu-central-1</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
</tr>
</tbody>
</table>

This diagram shows the general architecture:
For more information about Amazon ECS with AWS Fargate, see AWS Fargate on Amazon ECS (p. 30).

**EC2 Launch Type**

The EC2 launch type allows you to run your containerized applications on a cluster of Amazon EC2 instances that you manage.

This diagram shows the general architecture:
Using Data Volumes in Tasks

There are several use cases for using data volumes in Amazon ECS task definitions. We give the following guidance, broken down by launch type.

Fargate tasks only support nonpersistent storage volumes. For more information, see Fargate Task Storage (p. 230).

For EC2 tasks, use data volumes in the following common examples:
• To provide persistent data volumes for use with a container
• To define an empty, nonpersistent data volume and mount it on multiple containers
• To share defined data volumes at different locations on different containers on the same container instance
• To provide a data volume to your task that is managed by a third-party volume driver

The lifecycle of the volume can be tied to either a specific task or to the lifecycle of a specific container instance.

The following are the types of data volumes that can be used:

• Docker volumes — A Docker-managed volume that is created under /var/lib/docker/volumes on the container 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 only supported when using the EC2 launch type. Windows containers only support the use of the local driver. To use Docker volumes, specify a dockerVolumeConfiguration in your task definition. For more information, see Using volumes.

• Bind mounts — A file or directory on the host machine is mounted into a container. Bind mount host volumes are supported when using either the EC2 or Fargate launch types. To use bind mount host volumes, specify a host and optional sourcePath value in your task definition. For more information, see Using bind mounts.

Note
Before the release of the Amazon ECS-optimized AMI version 2017.03.a, only file systems that were available when the Docker daemon was started are available to Docker containers. You can use the latest Amazon ECS-optimized AMI to avoid this limitation, or you can upgrade the docker package to the latest version and restart Docker.

Topics
• Fargate Task Storage (p. 230)
• Docker Volumes (p. 231)
• Bind Mounts (p. 236)

Fargate Task Storage

When provisioned, each Fargate task receives the following storage. Task storage is ephemeral. After a Fargate task stops, the storage is deleted.

• 10 GB of Docker layer storage
• An additional 4 GB for volume mounts. This can be mounted and shared among containers using the volumes, mountPoints and volumesFrom parameters in the task definition.

Note
The host and sourcePath parameters are not supported for Fargate tasks.

For more information about Amazon ECS default service limits, see Amazon ECS Service Limits (p. 587).

To provide nonpersistent empty storage for containers in a Fargate task

In this example, you may have two database containers that need to access the same scratch file storage location during a task.

1. In the task definition volumes section, define a volume with the name database_scratch.
2. In the containerDefinitions section, create the database container definitions so they mount the nonpersistent storage.

```json
"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"
      }
    ]
  }
]
```

### Docker Volumes

When using Docker volumes, the built-in local driver or a third-party volume driver can be used. If a third-party driver is used, it should be installed on the container instance before the task is launched. 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.

Docker volumes are only supported when using the EC2 launch type. Windows containers only support the use of the local driver. To use Docker volumes, specify a dockerVolumeConfiguration in your task definition. For more information, see [Using Volumes](#using-volumes).

Some common use cases for Docker volumes are:

- 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 is managed by a third-party driver
Specifying a Docker Volume in your 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 task definition JSON shown below shows the syntax for the `volumes` and `mountPoints` objects for a container.

```json
{
    "containerDefinitions": [
        {
            "mountPoints": [
                {
                    "sourceVolume": "string",
                    "containerPath": "/path/to/mount_volume",
                    "readOnly": boolean
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "string",
            "dockerVolumeConfiguration": {
                "scope": "string",
                "autopropagation": 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 container definition `mountPoints`.

**dockerVolumeConfiguration**

Type: Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are only supported when using the EC2 launch type. Windows containers only support 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.

autoprov

Type: Boolean

Default value: false

Required: No

If this value is true, the Docker volume is created if it does not already exist.

Note
- This field is only used if the scope is shared.

driver

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because it is used for task placement. If the driver was installed using the Docker plugin CLI, use docker plugin ls to retrieve the driver name from your container instance. If the driver was installed using another method, use Docker plugin discovery to retrieve the driver name. For more information, see Docker plugin discovery. This parameter maps to Driver in the Create a volume section of the Docker Remote API and the --driver option to docker volume create.

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 the Docker Remote API and the --opt option to docker volume create.

labels

Type: String

Required: No

Custom metadata to add to your Docker volume. This parameter maps to Labels in the Create a volume section of the Docker Remote API and the --label option to docker volume create.

mountPoints

Type: Object

Required: No

The mount points for data volumes in your container.

This parameter maps to Volumes in the Create a container section of the Docker Remote 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 point cannot be across drives.

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 on the container to mount the volume at.

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.

Examples

The following are examples showing the use of Docker volumes.

**To provide nonpersistent storage for a container using a Docker volume**

In this example, you want a container to use an empty data volume that you aren't interested in keeping after the task has finished. For example, you may 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, set autoprovision to true so that the volume is created for use, and use the built-in local driver.

   "volumes": [
     {
       "name": "scratch",
       "dockerVolumeConfiguration": {
         "scope": "task",
         "autoprovision": true,
         "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.
To provide persistent storage for a container 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 using it has stopped. The built-in `local` driver is being used 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` so that the volume is created for use, and 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"  
      }  
    ]  
  }  
]
```
Bind Mounts

With bind mounts, a file or directory on the host machine is mounted into a container. Bind mount host volumes are supported when using either the EC2 or Fargate launch types. Fargate tasks only support nonpersistent storage volumes, so the host and sourcePath fields are not supported. For more information, see Fargate Task Storage (p. 230).

To use bind mount host volumes with tasks using the EC2 launch type, specify a host and optional sourcePath value in your task definition. For more information, see Using bind mounts.

Some common use cases for bind mounts are:

- To provide persistent data volumes for use with containers
- To define an empty, nonpersistent data volume and mount it on multiple containers on the same container instance
- To share defined data volumes at different locations on different containers on the same container instance

Specifying a Bind Mount in your Task Definition

Before your containers can use bind mount host 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 bind mount host volume, specify a host value and optional sourcePath value.

The following task definition JSON snippet shows the syntax for the volumes and mountPoints objects for a container:

```json
{
  "family": "",
  ...
  "containerDefinitions" : [
    {
      "mountPoints" : [
        {
          "containerPath" : "/path/to/mount_volume",
          "sourceVolume" : "string"
        },
        "name" : "string"
      ],
    },
    ...
    "volumes" : [
      {
        "host" : {
          "sourcePath" : "string"
        },
        "name" : "string"
      }
    ]
  }
}
```

name

Type: String
Bind Mounts

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 container definition mountPoints.

host

Required: No

This parameter is specified when using bind mounts. To use Docker volumes, specify a dockerVolumeConfiguration instead. The contents of the host parameter determine whether your bind mount data volume persists on the host container instance 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.

Bind mount host volumes are supported when using either the EC2 or Fargate launch types.

Windows containers can mount whole directories on the same drive as $env:ProgramData. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives. For example, you can mount C:\my\path:C:\my\path and D:\:D:\, but not D:\my\path:C:\my\path or D:\:C:\my\path.

sourcePath

Type: String

Required: No

When the host parameter is used, specify a sourcePath to declare the path on the host container instance that is presented to the container. If this parameter is empty, then the Docker daemon has assigned 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 container instance until you delete it manually. If the sourcePath value does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

mountPoints

Type: Object

Required: No

The mount points for data volumes in your container.

This parameter maps to Volumes in the Create a container section of the Docker Remote 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 point cannot be across drives.

sourceVolume

Type: String

Required: Yes, when mountPoints are used

The name of the volume to mount.

ccontainerPath

Type: String
Required: Yes, when mountPoints are used

The path on the container to mount the volume at.

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.

Examples

To provide nonpersistent empty storage for containers using a bind mount

In some cases, you want containers to share the same empty data volume, but you aren’t interested in keeping the data after the task has finished. For example, you may have two database containers that need to access the same scratch file storage location during a task. This task can be achieved using either a Docker volume or a bind mount host volume.

1. In the task definition volumes section, define a bind mount with the name database_scratch.

   Note
   Because the database_scratch bind mount does not specify a source path, 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 3 hours after the container exits, but you can configure this duration with the ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION agent variable). For more information, see Amazon ECS Container Agent Configuration (p. 152). If you need this data to persist, specify a sourcePath value for the bind mount.

   ```json
   "volumes": [
     {
       "name": "database_scratch",
       "host": {}
     }
   ]
   ``

2. In the containerDefinitions section, create the database container definitions so that they mount the nonpersistent storage.

   ```json
   "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 provide persistent storage for containers using a bind mount

When using bind mounts, if a `sourcePath` value is specified the data persists even after all containers that referenced it have stopped. Any files that exist at the `sourcePath` are presented to the containers at the `containerPath` value, and any files that are written to the `containerPath` value are written to the `sourcePath` value on the container instance.

**Important**

Amazon ECS does not sync your storage across container instances. Tasks that use persistent storage can be placed on any container instance in your cluster that has available capacity. If your tasks require persistent storage after stopping and restarting, you should always specify the same container instance at task launch time with the AWS CLI `start-task` command.

1. In the task definition `volumes` section, define a bind mount with `name` and `sourcePath` values.

```json
"volumes": [  
  {  
    "name": "webdata",  
    "host": {  
      "sourcePath": "/ecs/webdata"  
    }  
  }  
]
```

2. In the `containerDefinitions` section, define a container with `mountPoints` values that reference the name of the defined bind mount and the `containerPath` value to mount the bind mount at on the container.

```json
"containerDefinitions": [  
  {  
    "name": "web",  
    "image": "nginx",  
    "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

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, and you may 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.

```json
"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.

```json
"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

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, you need to build the image with the volume declared in a VOLUME instruction. The following example Dockerfile uses an httpd image and then adds a volume and mounts it at dockerfile_volume in the Apache document root (which 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 used in the following steps was built with the above 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 is also a host volume defined called host_etc, which 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 (which uses the image built with a volume in the Dockerfile) mounts the empty and host_etc volumes.

```
"containerDefinitions": [
  {
    "name": "web",
    "image": "my-repo/httpd_dockerfile_volume",
    "cpu": 100,
    "memory": 500,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "mountPoints": [
```

API Version 2014-11-13
241
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 (including the volume specified in the Dockerfile that was used to build the `my-repo/httpd_dockerfile_volume` image).

```json
{
  "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 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.

**Task Networking with the `awsvpc` Network Mode**

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.
Task networking also provides greater security for your containers by allowing you to use security groups and network monitoring tools at a more granular level within ECS tasks. Because each task gets its own elastic network interface, you can also take advantage of other Amazon EC2 networking features like VPC Flow Logs so that you can monitor traffic to and from your tasks. Additionally, containers that belong to the same task can communicate over the localhost interface. A task can only have one elastic network interface associated with it at a given time.

To use task networking, specify the awsvpc network mode in your task definition. Then, when you run a task or create a service, specify a network configuration that includes the subnets in which to place your tasks and the security groups to attach to its associated elastic network interface. The tasks are placed on valid container instances in the same Availability Zones as those subnets and the specified security groups are associated with the elastic network interface that is provisioned for the task.

The elastic network interface that is created for your task is fully managed by Amazon ECS. Amazon ECS creates the elastic network interface and attaches it to the container instance with the specified security group. The task sends and receives network traffic on the elastic network interface in the same way that Amazon EC2 instances do with their primary network interfaces. These elastic network interfaces are visible in the Amazon EC2 console for your account, but they cannot be detached manually or modified by your account. This is to prevent accidental deletion of an elastic network interface that is associated with a running task. You can view the elastic network interface 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 elastic network interface is detached and deleted.

### Enabling Task Networking

Your Amazon ECS container instances require at least version 1.15.0 of the container agent to enable task networking. 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 (p. 146). If you are using an Amazon ECS-optimized AMI, your instance needs at least version 1.15.0-4 of the ecs-init package. If your container instances are launched from version 2017.09.a or later, then they contain the required versions of the container agent and ecs-init. For more information, see Amazon ECS-optimized AMIs (p. 62).

**Important**

Currently, only Linux variants of the Amazon ECS-optimized AMI, or other Amazon Linux variants with the ecs-init package, support task networking.

To use task networking, your task definitions must specify the awsvpc network mode. For more information, see Network Mode (p. 196). When you run tasks or create services using a task definition that specifies the awsvpc network mode, you specify a network configuration that contains the VPC subnets to be considered for placement and the security groups to attach to the task's elastic network interface.

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 (p. 421). 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
```

### Task Networking Considerations

There are several things to consider when using task networking.

- The awsvpc network mode does not provide task elastic network interfaces with public IP addresses for tasks that use the EC2 launch type. To access the internet, tasks that use the EC2 launch type must
be launched in a private subnet that is 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 using the private IP address or DNS hostname, or routed through a load balancer from within the VPC. Tasks launched within public subnets do not have outbound network access.

**Note**
The above limitation does not apply to tasks that use the Fargate launch type. You can configure these tasks to receive public IP addresses.

- Currently, only Linux variants of the Amazon ECS-optimized AMI, or other Amazon Linux variants with the *ecs-init* package, support task networking. Your Amazon ECS container instances require at least version 1.15.0 of the container agent to enable task networking. We recommend using the latest container agent version.

- Each task that uses the *awsvpc* network mode receives its own elastic network interface, which is attached to the container instance that hosts it. EC2 instances have a limit to the number of elastic network interfaces that can be attached to them, and the primary network interface counts as one. For example, a *c4.large* instance may have up to three elastic network interfaces attached to it. The primary network adapter for the instance counts as one, so you can attach two more elastic network interfaces to the instance. Because each *awsvpc* task requires an elastic network interface, you can only run two such tasks on this instance type. For more information about how many elastic network interfaces are supported per instance type, see IP Addresses Per Network Interface Per Instance Type in the Amazon EC2 User Guide for Linux Instances.

- 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.

- Amazon ECS only accounts for the elastic network interfaces that it attaches to your container instances for you. If you have attached elastic network interfaces to your container instances manually, then Amazon ECS could try to place a task on an instance without sufficient available network adapter attachments. In this case, the task would time out, move from PROVISIONING to DEPROVISIONING, and then to STOPPED. We recommend that you do not attach elastic network interfaces to your container instances manually.

- Container instances must be registered with the *ecs.capability.task-eni* to be considered for placement of tasks with the *awsvpc* network mode. Container instances running version 1.15.0-4 or later of *ecs-init* are registered with this attribute.

- The elastic network interfaces that are created and attached to your container instances cannot be detached manually or modified by your account. This is to prevent the accidental deletion of an elastic network interface that is associated with a running task. To release the elastic network interfaces for a task, stop the task.

- When a task is started with the *awsvpc* network mode, the Amazon ECS container agent creates an additional *pause* container when running a task before starting the containers in the task definition. It then configures the network namespace of the *pause* container by executing 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 elastic network interface, and they can communicate with each other over the *localhost* interface.

- Services with tasks that use the *awsvpc* network mode (for example, those with the Fargate launch type) only support Application Load Balancers and Network Load Balancers; Classic Load Balancers are not supported. Also, 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 elastic network interface, not an Amazon EC2 instance. For more information, see Service Load Balancing (p. 306).
Using the awslogs Log Driver

You can configure the containers in your tasks to send log information to CloudWatch Logs. If you are using the Fargate launch type for your tasks, this allows you to view the logs from your containers. If you are using the EC2 launch type, this enables you to 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. This topic helps you get started using the awslogs log driver in your task definitions.

Note

The type of information that is logged by your task’s containers depends mostly on their ENTRYPOINT command. By default, the logs that are captured show the command output that you would normally 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. For more information on 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 Using CloudWatch Logs with Container Instances (p. 114). For more information about CloudWatch Logs, see Monitoring Log Files in the Amazon CloudWatch User Guide.

Topics

- Enabling the awslogs Log Driver for Your Containers (p. 245)
- Creating Your Log Groups (p. 246)
- Available awslogs Log Driver Options (p. 247)
- Specifying a Log Configuration in your Task Definition (p. 248)
- Viewing awslogs Container Logs in CloudWatch Logs (p. 250)

Enabling the awslogs Log Driver for Your Containers

If you are using the Fargate launch type for your tasks, all you need to do to enable the awslogs log driver is add the required logConfiguration parameters to your task definition. This configures your Fargate tasks to send log information to Amazon CloudWatch Logs. For more information, see Specifying a Log Configuration in your Task Definition (p. 248).

If you are using the EC2 launch type for your tasks and want to enable the awslogs log driver, your Amazon ECS container instances require at least version 1.9.0 of the container agent. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 146).

Note

If you are not using the Amazon ECS-optimized AMI (with at least version 1.9.0-1 of the ecs-init package) for your container instances, you also need to specify that the awslogs logging driver is available on the container instance when you start the agent by using the following environment variable in your docker run statement or environment variable file. For more information, see Installing the Amazon ECS Container Agent (p. 133).

    ECS_AVAILABLE_LOGGING_DRIVERS='["json-file","awslogs"]'

Your Amazon ECS container instances also require logs:CreateLogStream and logs:PutLogEvents permission on the IAM role with which you launch your container instances. If you created your Amazon ECS container instance role before awslogs log driver support was enabled in Amazon ECS, then you might need to add this permission. If your container instances use the managed IAM policy for container instances, then your container instances should have the correct permissions. For information about
Creating Your Log Groups

The `awslogs` log driver can send log streams to existing log groups in CloudWatch Logs, but it cannot create log groups. Before you launch any tasks that use the `awslogs` log driver, you should ensure the log groups that you intend your containers to use are created. The console provides an auto-configure option. If you register your task definitions in the console and choose the `Auto-configure CloudWatch Logs` option, your log groups are created for you. Alternatively, you can manually create your log groups using the following steps.

As an example, you could have a task with a WordPress container (which uses the `awslogs-wordpress` log group) that is linked to a MySQL container (which uses the `awslogs-mysql` log group). The sections below show how to create these log groups with the AWS CLI and with the CloudWatch console.

Creating a Log Group with the AWS CLI

The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts. For more information, see the AWS Command Line Interface User Guide.

If you have a working installation of the AWS CLI, you can use it to create your log groups. The command below creates a log group called `awslogs-wordpress` in the `us-west-2` region. Run this command for each log group to create, replacing the log group name with your value and region name to the desired log destination.

```
aws logs create-log-group --log-group-name awslogs-wordpress --region us-west-2
```

Creating a Log Group with the CloudWatch Console

The following procedure creates a log group in the CloudWatch console.

To create a log group in the CloudWatch console

2. In the left navigation pane, choose Storage and Logging, then Log groups.
3. Choose Add log group.
4. In the log group name field, enter a name for your log group.
5. Choose Create log group.

Creating a Log Group with the CloudWatch Console

The following procedure creates a log group in the CloudWatch console.

To create a log group in the CloudWatch console

2. In the left navigation pane, choose **Logs**.
3. Choose **Actions, Create log group**.
4. For **Log Group Name**, enter the name of the log group to create.
5. Choose **Create log group** to finish.

**Available awslogs Log Driver Options**

The **awslogs** log driver supports the following options in Amazon ECS task definitions. For more information, see [CloudWatch Logs logging driver](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/monitoring-logs.html).

<table>
<thead>
<tr>
<th>Option</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
</table>
| **awslogs-create-group** | No       | Specify whether you want the log group automatically created. If this option is not specified, it defaults to false.  
**Note**  
Your IAM policy must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`. |
| **awslogs-datetime-format** | 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. Thus 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.  
This option always takes precedence if both `awslogs-datetime-format` and `awslogs-multiline-pattern` are configured.  
**Note**  
Multiline logging performs regular expression parsing and matching of all log messages, which may have a negative impact on logging performance. |
| **awslogs-region** | Yes      | Specify the region to which the **awslogs** log driver should send your Docker logs. You can choose to send all of your logs from clusters in different regions to a single region in CloudWatch Logs so that they are all visible in one location, or you can separate them by region for more granularity. Be sure that the specified log group exists in the region that you specify with this option. |
| **awslogs-group**  | Yes      | You must specify a log group to which the **awslogs** log driver sends its log streams. For more information, see [Creating Your Log Groups](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/monitoring-logs.html) (p. 246). |
| **awslogs-multiline-pattern** | No       | This option defines a multiline start pattern using a regular expression. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. Thus the matched line is the delimiter between log messages. |
This option is ignored if awslogs-datetime-format is also configured.

**Note**
Multiline logging performs regular expression parsing and matching of all log messages. This may have a negative impact on logging performance.

**awslogs-stream-prefix**

Required: Optional for the EC2 launch type, required for the Fargate launch type.

The awslogs-stream-prefix option allows you to associate a log stream with the specified prefix, the container name, and the ID of the Amazon ECS task to which the container belongs. 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 do not specify a prefix with this option, then the log stream is named after the container ID that is assigned by the Docker daemon on the container instance. Because it is 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 could use the service name as the prefix, which would allow you to 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 to which the container belongs.

You must specify a stream-prefix for your logs in order to have your logs appear in the Log pane when using the Amazon ECS console.

## Specifying a Log Configuration in your Task Definition

Before your containers can send logs to CloudWatch, you must specify the awslogs log driver for containers in your task definition. This section describes the log configuration for a container to use the awslogs log driver. For more information, see [Creating a Task Definition](p. 188).

The task definition JSON shown below has a logConfiguration object specified for each container; one for the WordPress container that sends logs to a log group called awslogs-wordpress, and one for a MySQL container that sends logs to a log group called awslogs-mysql. Both containers use the awslogs-example log stream prefix.

```json
{
    "containerDefinitions": [
    {
        "name": "wordpress",
        "links": ["mysql"],
        "image": "wordpress",
        "essential": true,
        "portMappings": [
            {
                "containerPort": 80,
                "hostPort": 80
            }
        ],
        "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
                "awslogs-group": "awslogs-wordpress",
                "awslogs-region": "us-west-2",
            }
        }
    }
}
In the Amazon ECS console, the log configuration for the wordpress container is specified as shown in the image below.

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 Tasks (p. 269) and Creating a Service (p. 340).
Viewing awslogs Container Logs in CloudWatch Logs

For tasks using the EC2 launch type, after your container instance role has the proper permissions to send logs to CloudWatch Logs, your container agents are updated to at least version 1.9.0, and you have configured and started a task with containers that use the awslogs log driver, your configured containers should be sending their log data to CloudWatch Logs. You can view and search these logs in the console.

To view your CloudWatch Logs data for a container from the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the Clusters page, select the cluster that contains the task to view.
3. On the Cluster: cluster_name page, choose Tasks and select the task to view.
4. On the Task: task_id page, expand the container view by choosing the arrow to the left of the container name.
5. In the Log Configuration section, choose View logs in CloudWatch, which opens the associated log stream in the CloudWatch console.

To view your CloudWatch Logs data in the CloudWatch console

2. In the left navigation pane, choose Logs.
3. Select a log group to view. You should see the log groups that you created in Creating Your Log Groups (p. 246).
4. Choose a log stream to view.

Private Registry Authentication for Tasks

Private registry authentication for tasks using AWS Secrets Manager enables you to store your credentials securely and then reference them in your container definition. This allows your tasks to use images from private repositories. This feature is supported by tasks using both the Fargate or EC2 launch types.
Important
If your task definition references an image stored in Amazon ECR, this topic does not apply.
For more information, see Using Amazon ECR Images with Amazon ECS in the Amazon Elastic
Container Registry User Guide.

For tasks using the EC2 launch type, 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 checking your
agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 146).

For tasks using the Fargate launch type, this feature requires platform version 1.2.0 or later. For
information, see AWS Fargate Platform Versions (p. 36).

Within your container definition, specify repositoryCredentials with the full ARN of the secret that
you created. The secret you reference can be from a different Region than the task using it, but must be
from within the same account.

Note
When using the Amazon ECS API, AWS CLI, or AWS SDK, if the secret exists in the same Region
as the task you are launching then you can use either the full ARN or name of the secret. When
using the AWS Management Console, the full ARN of the secret must be specified.

The following is a snippet of a task definition showing the required parameters:

```json
"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 using the EC2 launch type. For more information, see Private Registry Authentication for
Container Instances (p. 164).

Required IAM Permissions for Private Registry
Authentication

The Amazon ECS task execution role is required to use this feature. This allows the container agent to
pull the container image. For more information, see Amazon ECS Task Execution IAM Role (p. 418).

To provide access to the secrets that you create, manually add the following permissions as an inline
policy to the task execution role. For more information, see Adding and Removing IAM Policies.

- secretsmanager:GetSecretValue
- kms:Decrypt—Required only if your key uses a custom KMS key and not the default key. The ARN for
  your custom key should be added as a resource.

An example inline policy adding the permissions is shown below.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
```
Enabling Private Registry Authentication

To create a basic secret

Use AWS Secrets Manager to create a secret for your private registry credentials.

1. Open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose Store a new secret.
3. For Select secret type, choose Other type of secrets.
4. Select Plaintext and enter your private registry credentials using the following format:

   ```json
   {
     "username": "privateRegistryUsername",
     "password": "privateRegistryPassword"
   }
   ``

5. Choose Next.
6. For Secret name, type an optional path and name, such as production/MyAwesomeAppSecret or development/TestSecret, and choose Next. You can optionally add a description to help you remember the purpose of this secret later.

   The secret name must be ASCII letters, digits, or any of the following characters: / _ +=.@-

7. (Optional) At this point, you can configure rotation for your secret. For this procedure, leave it at Disable automatic rotation and choose Next.

   For information about how to configure rotation on new or existing secrets, see Rotating Your AWS Secrets Manager Secrets.
8. Review your settings, and then choose Store secret to save everything you entered as a new secret in Secrets Manager.

To create a task definition that uses private registry authentication

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Task Definitions.
3. On the Task Definitions page, choose Create new Task Definition.
4. On the Select launch type compatibility page, choose the launch type for your tasks and then Next step.

   Note
   This step only applies to regions that currently support Amazon ECS using AWS Fargate. For more information, see AWS Fargate on Amazon ECS (p. 30).

5. For Task Definition Name, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. For **Task execution role**, either select your existing task execution role or choose **Create new role** to have one created for you. This role authorizes Amazon ECS to pull private images for your task. For more information, see Required IAM Permissions for Private Registry Authentication (p. 252).

   **Important**
   If the **Task execution role** field does not appear, choose **Configure via JSON** and manually add the `executionRoleArn` field to specify your task execution role. The following shows the syntax:

   ```json
   "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"
   ```

7. For each container to create in your task definition, complete the following steps:
   a. In the **Container Definitions** section, choose **Add container**.
   b. For **Container name**, type a name for your container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
   c. For **Image**, type the image name or path to your private image. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
   d. Select the **Private repository authentication** option.
   e. For **Secrets manager ARN**, enter the full Amazon Resource Name (ARN) of the secret that you created earlier. The value must be between 20 and 2048 characters.
   f. Fill out the remaining required fields and any optional fields to use in your container definitions. More container definition parameters are available in the **Advanced container configuration** menu. For more information, see Task Definition Parameters (p. 195).
   g. Choose **Add**.
8. When your containers are added, choose **Create**.

### Specifying Sensitive Data

Amazon ECS enables you to inject 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. This feature is supported by tasks using both the EC2 and Fargate launch types.

Secrets can be exposed to a container in the following ways:

- To inject sensitive data into your containers as environment variables, use the `secrets` container definition parameter.
- To reference sensitive information in the log configuration of a container, use the `secretOptions` container definition parameter.

For tasks that use the Fargate launch type, this feature requires that your task use platform version 1.3.0 or later. For information, see AWS Fargate Platform Versions (p. 36).

For tasks that use the EC2 launch type, this feature requires that your container instance have version 1.22.0 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 (p. 146).

**Note**
This feature is not available in the GovCloud (US-East) region.

**Topics**
- Injecting Sensitive Data as an Environment Variable (p. 255)
Injecting Sensitive Data as an Environment Variable

Within your container definition, specify secrets with the name of the environment variable to set in the container and the full ARN of either the Secrets Manager secret or Systems Manager Parameter Store parameter containing the sensitive data to present to the container. The parameter that you reference must be from within the same account but can be from a different Region than the container using the parameter.

**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 the full ARN must be specified.

The following is a snippet of a task definition showing the format when referencing an Secrets Manager secret.

```json
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
    }]
  }
}
```

The following is a snippet of a task definition showing the format when referencing an Systems Manager Parameter Store parameter.

```json
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
    }]
  }
}
```

Injecting 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 either the Secrets Manager secret or Systems Manager Parameter Store parameter containing the sensitive data to present to the container. The parameter that you reference must be from within the same account but can be from a different Region than the container using the parameter.

**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 the full ARN must be specified.
The following is a snippet of a task definition showing the format when referencing an Secrets Manager secret.

```json
{
"containerDefinitions": [{
"logConfiguration": [{
"logDriver": "splunk",
"options": {
"splunk-url": "https://cloud.splunk.com:8080"
},
"secretOptions": [{
"name": "splunk-token",
"valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
}]
}]
}
}
```

The following is a snippet of a task definition showing the format when referencing an Systems Manager Parameter Store parameter.

```json
{
"containerDefinitions": [{
"logConfiguration": [{
"logDriver": "fluentd",
"options": {
"tag": "fluentd demo"
},
"secretOptions": [{
"name": "fluentd-address",
"valueFrom": "arn:aws:ssm:region:aws_account_id:parameter:parameter_name"
}]
}]
}
}
```

### Required IAM Permissions for Amazon ECS Secrets

To use this feature, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary AWS Systems Manager or Secrets Manager resources. For more information, see Amazon ECS Task Execution IAM Role (p. 418).

To provide access to the AWS Systems Manager Parameter Store parameters that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see Adding and Removing IAM Policies.

- `ssm:GetParameters`—Required if you are referencing a Systems Manager Parameter Store parameter in a task definition.
- `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.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions:

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["ssm:GetParameters"]
  },
  {
    "Effect": "Allow",
    "Action": ["secretsmanager:GetSecretValue"]
  },
  {
    "Effect": "Allow",
    "Action": ["kms:Decrypt"]
  }]
}
```
Creating an AWS Secrets Manager Secret

You can use the Secrets Manager console to create a secret for your sensitive data. For more information, see Creating a Basic Secret in the AWS Secrets Manager User Guide.

To create a basic secret

Use Secrets Manager to create a secret for your sensitive data.

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose Store a new secret.
3. For Select secret type, choose Other type of secrets.
4. Specify the details of your custom secret as Key and Value pairs. For example, you can specify a key of UserName, and then supply the appropriate user name as its value. Add a second key with the name of Password and the password text as its value. You could also add entries for Database name, Server address, TCP port, and so on. You can add as many pairs as you need to store the information you require.

   Alternatively, you can choose the Plaintext tab and enter the secret value in any way you like.

5. Choose the AWS KMS encryption key that you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks to see if there's a default key for the account, and uses it if it exists. If a default key doesn't exist, Secrets Manager creates one for you automatically. You can also choose Add new key to create a custom CMK specifically for this secret.

6. Choose Next.
7. For Secret name, type an optional path and name, such as production/MyAwesomeAppSecret or development/TestSecret, and choose Next. You can optionally add a description to help you remember the purpose of this secret later.

   The secret name must be ASCII letters, digits, or any of the following characters: /_+=.@-

8. (Optional) At this point, you can configure rotation for your secret. For this procedure, leave it at Disable automatic rotation and choose Next.

   For information about how to configure rotation on new or existing secrets, see Rotating Your AWS Secrets Manager Secrets.

9. Review your settings, and then choose Store secret to save everything you entered as a new secret in Secrets Manager.
Creating an AWS Systems Manager Parameter Store Parameter

You can use the AWS Systems Manager console to create a Systems Manager Parameter Store parameter for your sensitive data. For more information, see Walkthrough: Create and Use a Parameter in a Command (Console) in the AWS Systems Manager User Guide.

To create a Parameter Store parameter

2. In the navigation pane, choose Parameter Store, Create parameter.
3. For Name, type a hierarchy and a parameter name. For example, type test/database_password.
   
   Note
   If you are referencing an AWS Secrets Manager secret in your parameter, the parameter name must begin with the following reserved path: /aws/reference/secretsmanager/. For more information, see Referencing AWS Secrets Manager Secrets from Parameter Store Parameters in the AWS Systems Manager User Guide.

4. For Description, type an optional description.
5. For Type, choose String, StringList, or SecureString.
   
   Note
   • If you choose SecureString, the KMS Key ID field appears. If you don't provide a KMS CMK ID, a KMS CMK ARN, an alias name, or an alias ARN, then the system uses alias/aws/ssm, which is the default KMS CMK for Systems Manager. To avoid using this key, choose a custom key. For more information, see Use Secure String Parameters in the AWS Systems Manager User Guide.
   • When you create a secure string parameter in the console by using the key-id parameter with either a custom KMS CMK alias name or an alias ARN, you must specify the prefix alias/ before the alias. The following is an ARN example:

     ```
     arn:aws:kms:us-east-2:123456789012:alias/MyAliasName
     ```

   The following is an alias name example:

   ```
   alias/MyAliasName
   ```

6. For Value, type a value. For example, MyFirstParameter. If you chose SecureString, the value is masked as you type.
7. Choose Create parameter.

Creating a Task Definition that References a Secret

You can use the Amazon ECS console to create a task definition that references either a Secrets Manager secret or a Systems Manager Parameter Store parameter.

To create a task definition that specifies a secret

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Task Definitions, Create new Task Definition.
3. On the Select launch type compatibility page, choose the launch type for your tasks and choose Next step.
Creating a Task Definition that References a Secret

4. For **Task Definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

5. For **Task execution role**, either select your existing task execution role or choose **Create new role** to have one created for you. This role authorizes Amazon ECS to pull private images for your task. For more information, see **Required IAM Permissions for Private Registry Authentication** (p. 252).

   **Important**
   If the **Task execution role** field does not appear, choose **Configure via JSON** and manually add the `executionRoleArn` field to specify your task execution role. The following code shows the syntax:

   ```json
   "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"
   ```

6. For each container to create in your task definition, complete the following steps:
   a. Under **Container Definitions**, choose **Add container**.
   b. For **Container name**, type a name for your container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
   c. For **Image**, type the image name or path to your private image. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
   d. Expand **Advanced container configuration**.
   e. For container secrets referenced as environment variables, under **Environment**, for **Environment variables**, complete the following fields:
      i. For **Key**, enter the name of the environment variable to set in the container. This corresponds to the `name` field in the `secrets` section of a container definition.
      ii. For **Value**, choose **ValueFrom**. For **Add value**, enter the full ARN or the Secrets Manager secret or the name or full ARN of the AWS Systems Manager Parameter Store parameter that contains the data to present to your container as an environment variable.
         **Note**
         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 secret. If the parameter exists in a different Region, then the full ARN must be specified.
   f. For secrets referenced in the log configuration for a container, under **Storage and Logging**, for **Log configuration**, complete the following fields:
      i. Clear the **Auto-configure CloudWatch Logs** option.
      ii. Under **Log options**, for **Key**, enter the name of the log configuration option to set.
      iii. For **Value**, choose **ValueFrom**. For **Add value**, enter the full ARN or the Secrets Manager secret or the name or full ARN of the AWS Systems Manager Parameter Store parameter that contains the data to present to your log configuration as a log option.
         **Note**
         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 secret. If the parameter exists in a different Region, then the full ARN must be specified.
   g. Fill out the remaining required fields and any optional fields to use in your container definitions. More container definition parameters are available in the **Advanced container configuration** menu. For more information, see **Task Definition Parameters** (p. 195).
Example Task Definitions

This section provides some task definition examples that you can use to start creating your own task definitions. For more information, see Task Definition Parameters (p. 195) and Creating a Task Definition (p. 188).

Topics

- Example: Webserver (p. 260)
- Example: WordPress and MySQL (p. 261)
- Example: awlogs Log Driver (p. 262)
- Example: splunk Log Driver (p. 262)
- Example: fluentd Log Driver (p. 263)
- Example: gelf Log Driver (p. 263)
- Example: Amazon ECR Image and Task Definition IAM Role (p. 264)
- Example: Entrypoint with Command (p. 264)
- Example: Container Dependency (p. 264)

Example: Webserver

The following is an example task definition using the 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": "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"
            }
        ]
    }
}
```
Example: WordPress and MySQL

The following example specifies a WordPress container and a MySQL container that are linked together. This WordPress container exposes the container port 80 on the host port 80. The security group on the container instance would need to open port 80 in order for this WordPress installation to be accessible from a web browser.

For more information about the WordPress container, see the official WordPress Docker Hub repository at https://registry.hub.docker.com/_/wordpress/. For more information about the MySQL container, go to the official MySQL Docker Hub repository at https://registry.hub.docker.com/_/mysql/.

```json
{
  "containerDefinitions": [
    {
      "name": "wordpress",
      "links": ["mysql"],
      "image": "wordpress",
      "essential": true,
      "memory": 500,
      "cpu": 10
    },
    {
      "environment": [
        {
          "name": "MYSQL_ROOT_PASSWORD",
          "value": "password"
        }
      ],
      "name": "mysql",
      "image": "mysql",
      "cpu": 10,
      "memory": 500,
      "essential": true
    }
  ],
  "family": "hello_world"
}
```

**Important**

If you use this task definition with a load balancer, you need to complete the WordPress setup installation through the web interface on the container instance immediately after the
container starts. The load balancer health check ping expects a 200 response from the server, but WordPress returns a 301 until the installation is completed. If the load balancer health check fails, the load balancer deregisters the instance.

Example: awslogs Log Driver

The following example demonstrates how to use the awslogs log driver in a task definition that uses the Fargate launch type. The nginx container sends its logs to the ecs-log-streaming log group in the us-west-2 region. For more information, see Using the awslogs Log Driver (p. 245).

```json
{
    "containerDefinitions": [
        {
            "memory": 128,
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "name": "nginx-container",
            "image": "nginx",
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "ecs-log-streaming",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "fargate-task-1"
                }
            }
        },
        "cpu": 0
    ],
    "networkMode": "awsvpc",
    "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
    "memory": "2048",
    "cpu": "1024",
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "family": "example_task_1"
}
```

Example: splunk Log Driver

The following example 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 Specifying Sensitive Data (p. 254).

```json
"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-KrnBkD"
        }
    }
]
```
Example: fluentd Log Driver

The following example 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 Specifying Sensitive Data (p. 254).

```
"containerDefinitions": [{
  "logConfiguration": {
    "logDriver": "fluentd",
    "options": {
      "tag": "fluentd demo"
    },
    "secretOptions": [{
      "name": "fluentd-address",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:fluentd-address-KnzBkD"
    }]
  },
  "entryPoint": [],
  "portMappings": [{
    "hostPort": 80,
    "protocol": "tcp",
    "containerPort": 80
  },
  {"hostPort": 24224,
    "protocol": "tcp",
    "containerPort": 24224
  }
}],
```

Example: gelf Log Driver

The following example 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 (p. 209).

```
"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
  }
}],
```
Example: Amazon ECR Image and Task Definition IAM Role

The following example 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 IAM Roles for Tasks (p. 435).

```json
{
    "containerDefinitions": [
        {
            "name": "sample-app",
            "memory": 200,
            "cpu": 10,
            "essential": true
        }
    ],
    "family": "example_task_3",
    "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
}
```

Example: Entrypoint with Command

The following example demonstrates the syntax for a Docker container that uses an entry point and a command argument. This container pings `google.com` four times and then exits.

```json
{
    "containerDefinitions": [
        {
            "memory": 32,
            "essential": true,
            "entryPoint": [
                "ping"
            ],
            "name": "alpine_ping",
            "readonlyRootFilesystem": true,
            "image": "alpine:3.4",
            "command": [
                "-c",
                "4",
                "google.com"
            ],
            "cpu": 16
        }
    ],
    "family": "example_task_2"
}
```

Example: Container Dependency

This example 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 healthcheck parameters, before the `app` container will start. For more information, see Container Dependency (p. 216).

```json
{

```
Example: Container Dependency

```json
"family": "appmesh-gateway",
"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": "111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-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" ] }
    }
  ]
}
```
Updating a Task Definition

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.

To create a task definition revision

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
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, select the box to the left of the task definition to revise and choose Create new revision.
5. On the Create new revision of Task Definition page, make changes. For example, to change the existing container definitions (such as the container image, memory limits, or port mappings), select the container, make the changes, and then choose Update.
6. Verify the information and choose Create.
7. If your task definition is used in a service, update your service with the updated task definition. For more information, see Updating a Service (p. 350).

Deregistering Task Definitions

If you decide that you no longer need a task definition in Amazon ECS, you can deregister the task definition 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, it is immediately marked as INACTIVE. Existing tasks and services that reference an INACTIVE task definition continue to run without disruption, and existing services that reference an INACTIVE task definition can still scale up or down by modifying the service's desired count.

You cannot use an INACTIVE task definition to run new tasks or create new services, and you cannot update an existing service to reference an INACTIVE task definition (although there may be up to a 10-minute window following deregistration where these restrictions have not yet taken effect).

Note
At this time, INACTIVE task definitions remain discoverable in your account indefinitely; however, this behavior is subject to change in the future, so you should not rely on INACTIVE task definitions persisting beyond the lifecycle of any associated tasks and services.

Use the following procedure to deregister a task definition.

To deregister a task definition

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
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 name that contains one or more revisions that you want to deregister.
5. On the Task Definition Name page, select the box to the left of each task definition revision you want to deregister.
6. Choose Actions, Deregister.
7. Verify the information in the Deregister Task Definition window, and choose Deregister to finish.
Scheduling Amazon ECS Tasks

Amazon Elastic Container Service (Amazon ECS) is a shared state, optimistic concurrency system that provides flexible scheduling capabilities for your tasks and containers. The Amazon ECS schedulers leverage the same cluster state information provided by the Amazon ECS API to make appropriate placement decisions.

Each task that uses the Fargate launch type has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

Amazon ECS provides a service scheduler (for long-running tasks and applications), the ability to run tasks manually (for batch jobs or single run tasks), with Amazon ECS placing tasks on your cluster for you. You can specify task placement strategies and constraints that allow you to run tasks in the configuration you choose, such as spread out across Availability Zones. It is also possible to integrate with custom or third-party schedulers.

Service Scheduler

The service scheduler is ideally suited for long running stateless services and applications. The service scheduler ensures that the scheduling strategy you specify is followed and reschedules tasks when a task fails (for example, if the underlying infrastructure fails for some reason).

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 (p. 291).

- 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 (p. 291).

**Note**

Fargate tasks do not support the DAEMON scheduling strategy.

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, such as deploying a new task definition, or changing the running number of desired tasks. By default, the service scheduler spreads tasks across Availability Zones, but you can use task placement strategies and constraints to customize task placement decisions. For more information, see Services (p. 290).

Manually Running Tasks

The RunTask action is ideally suited for processes such as batch jobs that perform work and then stop. For example, you could 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, which minimizes the chances that a single instance gets a disproportionate number of tasks. Alternatively, you can use RunTask to customize how the scheduler places tasks using task placement strategies and constraints. For more information, see Running Tasks (p. 269) and RunTask in the Amazon Elastic Container Service API Reference.

Running Tasks on a `crontab`-like Schedule
If you have tasks to run at set intervals in your cluster, such as a backup operation or a log scan, you can use the Amazon ECS console to create a CloudWatch Events rule that runs one or more tasks in your cluster at specified times. Your scheduled event rule can be set to either a specific interval (run every \( N \) minutes, hours, or days), or for more complicated scheduling, you can use a cron expression. For more information, see Scheduled Tasks (cron) (p. 283).

**Custom Schedulers**

Amazon ECS allows you to create your own schedulers that meet the needs of your business, or to leverage third party schedulers. **Blox** is an open-source project that gives you more control over how your containerized applications run on Amazon ECS. It enables you to build schedulers and integrate third-party schedulers with Amazon ECS while leveraging Amazon ECS to fully manage and scale your clusters. Custom schedulers use the StartTask API operation to place tasks on specific container instances within your cluster.

**Note**

Custom schedulers are only compatible with tasks using the EC2 launch type. If you are using the Fargate launch type for your tasks, the StartTask API does not work.

**Task Placement**

The RunTask and CreateService actions enable you to specify task placement constraints and task placement strategies to customize how Amazon ECS places your tasks. For more information, see Amazon ECS Task Placement (p. 274).

**Contents**

- Running Tasks (p. 269)
- Amazon ECS Task Placement (p. 274)
- Scheduled Tasks (cron) (p. 283)
- Task Lifecycle (p. 285)
- Task Retirement (p. 287)
- Fargate Task Recycling (p. 288)
- Creating a Scheduled Task Using the AWS CLI (p. 289)

**Running Tasks**

Running tasks manually is ideal in certain situations. For example, suppose that you are developing a task but you are not ready to deploy this task with the service scheduler. Perhaps your task is a one-time or periodic batch job that does not make sense to keep running or restart when it finishes.

To keep a specified number of tasks running or to place your tasks behind a load balancer, use the Amazon ECS service scheduler instead. For more information, see Services (p. 290).

**Contents**

- Running a Task Using the Fargate Launch Type (p. 269)
- Running a Task Using the EC2 Launch Type (p. 271)

**Running a Task Using the Fargate Launch Type**

To run a task using the Fargate launch type, do the following:

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose **Task Definitions** and select the task definition to run.
   
   • To run the latest revision of a task definition shown here, select the box to the left of the task definition to run.
   • To run an earlier revision of a task definition shown here, select the task definition to view all active revisions, then select the revision to run.

3. Choose **Actions, Run Task**.

4. In the **Run Task** section, complete the following steps:
   
   a. For **Launch type**, choose **FARGATE**. For more information about launch types, see Amazon ECS Launch Types (p. 227).
   b. For **Platform version**, choose **LATEST**. For more information about platform versions, see AWS Fargate Platform Versions (p. 36).
   c. For **Cluster**, choose the cluster to use.
   d. For **Number of tasks**, type the number of tasks to launch with this task definition.
   e. For **Task Group**, type the name of the task group.

5. In the **VPC and security groups** section, complete the following steps:
   
   a. For **Cluster VPC**, choose the VPC for your tasks to use. Ensure that the VPC that you choose is not configured to require dedicated hardware tenancy, as that is not supported by Fargate tasks.
   b. For **Subnets**, choose the available subnets for your task.
   c. For **Security groups**, a security group has been created for your task that allows HTTP traffic from the internet (0.0.0.0/0). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.
   d. For **Auto-assign public IP**, choose **ENABLED** if you want the elastic network interface attached to the Fargate task to be assigned a public IP address. This is required if your task needs outbound network access, for example to pull an image. If outbound network access is not required, then you can choose **DISABLED**.

6. In the **Advanced Options** section, complete the following steps:
   
   • (Optional) To send command or environment variable overrides to one or more containers in your task definition, or to specify an IAM role task override, choose **Advanced Options** and complete the following steps:
     
     i. For **Task Role Override**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 435).

        Only roles with the Amazon EC2 Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 437).

     ii. For **Task Execution Role Override**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 435).

        Only roles with the Amazon EC2 Container Service Task Execution Role trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 437).

     iii. For **Container Overrides**, choose a container to which to send a command or environment variable override.

        • **For a command override**: For **Command override**, type the command override to send. If your container definition does not specify an **ENTRYPOINT**, the format should be a comma-separated list of non-quoted strings. For example:
Running a Task Using the EC2 Launch Type

To run a task using the EC2 launch type, do the following:

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Task Definitions and select the task definition to run.
To run the latest revision of a task definition shown here, select the box to the left of the task definition to run.

To run an earlier revision of a task definition shown here, select the task definition to view all active revisions, then select the revision to run.

3. Choose **Actions, Run Task**.
4. For **Launch Type**, choose **EC2**. For more information about launch types, see Amazon ECS Launch Types (p. 227).
5. For **Cluster**, choose the cluster to use. For **Number of tasks**, type the number of tasks to launch with this task definition. For **Task Group**, type the name of the task group.
6. If your task definition uses the `awsvpc` network mode, complete these substeps. Otherwise, continue to the next step.
   a. For **Cluster VPC**, choose the VPC that your container instances reside in.
   b. For **Subnets**, choose the available subnets for your task.
      
      **Important**
      Only private subnets are supported for the `awsvpc` network mode. Because tasks do not receive public IP addresses, a NAT gateway is required for outbound internet access, and inbound internet traffic should be routed through a load balancer.
   c. For **Security groups**, a security group has been created for your task that allows HTTP traffic from the internet (0.0.0.0/0). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.
7. (Optional) For **Task Placement**, you can specify how tasks are placed using task placement strategies and constraints. Choose from the following options:
   a. **AZ Balanced Spread** - distribute tasks across Availability Zones and across container instances in the Availability Zone.
   b. **AZ Balanced BinPack** - distribute tasks across Availability Zones and across container instances with the least available memory.
   c. **BinPack** - distribute tasks based on the least available amount of CPU or memory.
   d. **One Task Per Host** - place, at most, one task from the service on each container instance.
   e. **Custom** - define your own task placement strategy. See Amazon ECS Task Placement (p. 274) for examples.

For more information, see Amazon ECS Task Placement (p. 274).
8. (Optional) To send command or environment variable overrides to one or more containers in your task definition, or to specify an IAM role task override, choose **Advanced Options** and complete the following steps:
   a. For **Task Role Override**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 435).
      
      Only roles with the Amazon EC2 Container Service Task Role trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 437).
   b. For **Task Execution Role Override**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see IAM Roles for Tasks (p. 435).
      
      Only roles with the Amazon EC2 Container Service Task Execution Role trust relationship are shown here. For more information about creating an IAM role for your tasks, see Creating an IAM Role and Policy for your Tasks (p. 437).
c. For **Container Overrides**, choose a container to which to send a command or environment variable override.

- **For a command override:** For **Command override**, type the command override to send. If your container definition does not specify an `ENTRYPOINT`, the format should be a comma-separated list of non-quoted strings. For example:

  ```bash
  /bin/sh,-c,echo,$DATE
  ```

  If your container definition does specify an `ENTRYPOINT` (such as `sh,-c`), the format should be an unquoted string, which is surrounded with double quotes and passed as an argument to the `ENTRYPOINT` command. For example:

  ```bash
  while true; do echo $DATE > /var/www/html/index.html; sleep 1; done
  ```

- **For environment variable overrides:** Choose **Add Environment Variable**. For **Key**, type the name of your environment variable. For **Value**, type a string value for your environment value (without surrounding quotes).

  ![Add Environment Variable](image)

  This environment variable override is sent to the container as:

  ```bash
  MY_ENV_VAR="This variable contains a string."
  ```

9. In the **Task tagging configuration** section, complete the following steps:

   a. Select **Enable ECS managed tags** if you want Amazon ECS to automatically tag each task with the Amazon ECS managed tags. For more information, see Tagging Your Amazon ECS Resources.

   b. For **Propagate tags from**, select one of the following:

      - **Do not propagate** – This option will not propagate any tags.
      - **Task Definitions** – This option will propagate the tags specified in the task definition to the task.

      **Note**
      If you specify a tag with the same key in the **Tags** section, it will override the tag propagated from the task definition.

10. In the **Tags** section, specify the key and value for each tag to associate with the task. For more information, see Tagging Your Amazon ECS Resources.

11. Review your task information and choose **Run Task**.

**Note**
If your task moves from **PENDING** to **STOPPED**, or if it displays a **PENDING** status and then disappears from the listed tasks, your task may be stopping due to an error. For more information, see Checking Stopped Tasks for Errors (p. 590) in the troubleshooting section.
Amazon ECS Task Placement

When a task that uses the EC2 launch type is launched, 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. Task placement strategies and constraints are not supported for tasks using the Fargate launch type. By default, Fargate tasks are spread across Availability Zones.

A task placement strategy is an algorithm for selecting instances for task placement or tasks for termination. For example, Amazon ECS can select instances at random or it can select instances such that tasks are distributed evenly across a group of instances.

A task placement constraint is a rule that is considered during task placement. For example, you can use constraints to place tasks based on Availability Zone or instance type. You can also associate attributes, which are name/value pairs, with your container instances and then use a constraint to place tasks based on attribute.

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 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 instances that satisfy the CPU, memory, and port requirements in the task definition.
2. Identify the instances that satisfy the task placement constraints.
3. Identify the instances that satisfy the task placement strategies.
4. Select the instances for task placement.

Contents
- Amazon ECS Task Placement Strategies (p. 274)
- Amazon ECS Task Placement Constraints (p. 276)
- Cluster Query Language (p. 280)

Amazon ECS Task Placement Strategies

A task placement strategy is an algorithm for selecting instances for task placement or tasks for termination. Task placement strategies can be specified when either running a task or creating a new service. For more information, see Amazon ECS Task Placement (p. 274).

Strategy Types

Amazon ECS supports the following task placement strategies:

binpack

Place tasks based on the least available amount of CPU or memory. This minimizes the number of instances in use.
random

Place tasks randomly.

spread

Place tasks evenly based on the specified value. Accepted values are attribute key-value pairs, instanceId, or host. Service tasks are spread based on the tasks from that service.

Example Strategies

You can specify task placement strategies with the following actions: CreateService and RunTask.

The following strategy distributes tasks evenly across Availability Zones.

```
"placementStrategy": [
  {
    "field": "attribute:ecs.availability-zone",
    "type": "spread"
  }
]
```

The following strategy distributes tasks evenly across all instances.

```
"placementStrategy": [
  {
    "field": "instanceId",
    "type": "spread"
  }
]
```

The following strategy bin packs tasks based on memory.

```
"placementStrategy": [
  {
    "field": "memory",
    "type": "binpack"
  }
]
```

The following strategy places tasks randomly.

```
"placementStrategy": [
  {
    "type": "random"
  }
]
```

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",
```
The following strategy distributes tasks evenly across Availability Zones and then bin packs tasks based on memory within each Availability Zone.

```json
"placementStrategy": [
  {
    "field": "attribute:ecs.availability-zone",
    "type": "spread"
  },
  {
    "field": "memory",
    "type": "binpack"
  }
]
```

Amazon ECS Task Placement Constraints

A task placement constraint is a rule that is considered during task placement. For more information, see Amazon ECS Task Placement (p. 274).

**Constraint Types**

Amazon ECS supports the following types of task placement constraints:

- **distinctInstance**
  Place each task on a different container instance. This task placement constraint can be specified when either running a task or creating a new service.

- **memberOf**
  Place tasks on container instances that satisfy an expression. For more information about the expression syntax for constraints, see Cluster Query Language (p. 280).

  The `memberOf` task placement constraint can be specified with the following actions:
  - Running a task
  - Creating a new service
  - Creating a new task definition
  - Creating a new revision of an existing task definition

**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.

**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-eca289fb".

API Version 2014-11-13
276
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".

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".

Adding an Attribute

You can add custom attributes at instance registration time using the container agent or manually, using the AWS Management Console. For more information about using the container agent, see Amazon ECS Container Agent Configuration Parameters (p. 159).

To add custom attributes using the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Clusters and select a cluster.
3. On the ECS Instances tab, select the check box for the container instance.
5. For each attribute, do the following:
   a. Choose Add attribute.
   b. Type a name and a value for the attribute and choose the checkmark icon.
6. When you are finished adding attributes, choose Close.

Adding custom attributes using the AWS CLI

The following examples demonstrate how to add custom attributes using the put-attributes command.

Example: Single Attribute

The following example adds the custom attribute "stack=prod" to the specified container instance in the default cluster.

```
aws ecs put-attributes --attributes name=stack,value=prod,targetId=arn
```

Example: Multiple Attributes

The following example adds the custom attributes "stack=prod" and "project=a" to the specified container instance in the default cluster.

```
aws ecs put-attributes --attributes name=stack,value=prod,targetId=arn
name=project,value=a,targetId=arn
```
Filtering by Attribute

You can apply a filter for your container instances, allowing you to see custom attributes.

Filter container instances by attribute using the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose a cluster that has container instances.
3. Choose **ECS Instances**.
4. Set column visibility preferences by choosing the gear icon (⚙️) and selecting the attributes to display. This setting persists across all container clusters associated with your account.
5. Using the **Filter by attributes** text field, type or select the attributes you would like to filter by. The format must be `AttributeName:AttributeValue`.

   For **Filter by attributes**, type or select the attributes by which to filter. After you select the attribute name, you are prompted for the attribute value.
6. Add additional attributes to the filter as needed. Remove an attribute by choosing the X next to it.

Filter container instances by attribute using the AWS CLI

The following examples demonstrate how to filter container instances by attribute using the `list-container-instances` command. For more information about the filter syntax, see [Cluster Query Language (p. 280)](#).

**Example: Built-in Attribute**

The following example uses built-in attributes to list the g2.2xlarge instances.

```bash
aws ecs list-container-instances --filter "attribute:ecs.instance-type == g2.2xlarge"
```

**Example: Custom Attribute**

The following example lists the instances with the custom attribute "stack=prod".

```bash
aws ecs list-container-instances --filter "attribute:stack == prod"
```

**Example: Exclude an Attribute Value**

The following example lists the instances with the custom attribute "stack" unless the attribute value is "prod".

```bash
aws ecs list-container-instances --filter "attribute:stack != prod"
```

**Example: Multiple Attribute Values**

The following example uses built-in attributes to list the instances of type `t2.small` or `t2.medium`.

```bash
aws ecs list-container-instances --filter "attribute:ecs.instance-type in [t2.small, t2.medium]"
```

**Example: Multiple Attributes**

The following example uses built-in attributes to list the T2 instances in the us-east-1a Availability Zone.
Task Groups

You can identify a set of related tasks as a task group. All tasks with the same task group name are considered as a set when performing spread placement. For example, suppose that you are 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 this task group as a constraint for task placement.

When you launch a task using the RunTask or StartTask action, you can specify the name of the task group for the task. If you don't specify a task group for the task, the default name is the family name of the task definition (for example, family:my-task-definition).

For tasks launched by the service scheduler, the task group name is the name of the service (for example, service:my-service-name).

Limits

- A task group name must be 255 characters or less.
- Each task can be in exactly one group.
- After launching a task, you cannot modify its task group.

Example 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, RegisterTaskDefinition, and RunTask.

```
"placementConstraints": [ 
  { 
    "expression": "attribute:ecs.instance-type =~ t2.*", 
    "type": "memberOf"
  } 
]
```

The example uses the memberOf constraint to place tasks on instances in the databases task group. It can be specified with the following actions: CreateService, 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 and RunTask.

```
"placementConstraints": [ 
  { 
    "type": "distinctInstance"
  } 
]
```
Cluster Query Language

Cluster queries are expressions that enable 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 Attributes (p. 276).

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 Tasks (p. 269) and Creating a Service (p. 340). You can also apply a group filter when listing container instances. For more information, see Filtering by Attribute (p. 278).

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 (lin), 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 Attributes (p. 276).

dc2InstanceId

Select container instances by their Amazon EC2 instance ID.

Valid operators: equals (==), not_equals (!=), in, not_in (lin), 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 (<=)


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 Task Groups (p. 279).

Operator

The comparison operator. The following operators are supported.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==, equals</td>
<td>String equality</td>
</tr>
<tr>
<td>!=, not_equal</td>
<td>String inequality</td>
</tr>
<tr>
<td>&gt;, greater_than</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=, greater_equal</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;, less_than</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=, less_equal</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>exists</td>
<td>Subject exists</td>
</tr>
<tr>
<td>!exists, not_exists</td>
<td>Subject does not exist</td>
</tr>
<tr>
<td>in</td>
<td>Value in argument list</td>
</tr>
<tr>
<td>!in, not_in</td>
<td>Value not in argument list</td>
</tr>
<tr>
<td>=~, matches</td>
<td>Pattern match</td>
</tr>
<tr>
<td>!~, not_matches</td>
<td>Pattern mismatch</td>
</tr>
</tbody>
</table>

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:

\[(\text{expression1 or expression2}) \text{ and expression3}\]

**Example Expressions**

The following are example expressions.

**Example: String Equality**

The following expression selects instances with the specified instance type.

\[\text{attribute:ecs.instance-type} \text{ == t2.small}\]

**Example: Argument List**

The following expression selects instances in the us-east-1a or us-east-1b Availability Zone.

\[\text{attribute:ecs.availability-zone in [us-east-1a, us-east-1b]}\]

**Example: Compound Expression**

The following expression selects G2 instances that are not in the us-east-1d Availability Zone.

\[\text{attribute:ecs.instance-type} \text{ =~ g2.* and attribute:ecs.availability-zone} \text{ != us-east-1d}\]

**Example: Task Affinity**

The following expression selects instances that are hosting tasks in the service:production group.

\[\text{task:group} \text{ == service:production}\]

**Example: Task Anti-Affinity**

The following expression selects instances that are not hosting tasks in the database group.

\[\text{not(task:group} \text{ == database)}\]

**Example: Running task count**

The following expression selects instances that are only running one task.

\[\text{runningTasksCount} \text{ == 1}\]

**Example: Amazon ECS container agent version**

The following expression selects instances that are running a container agent version below 1.14.5.

\[\text{agentVersion} \text{ < 1.14.5}\]

**Example: Instance registration time**

The following expression selects instances that were registered before February 13, 2018.

\[\text{registeredAt} \text{ < 2018-02-13}\]
Scheduled Tasks (cron)

You can run Amazon ECS tasks on a cron-like schedule using CloudWatch Events rules and targets.

If you have tasks to run at set intervals in your cluster, such as a backup operation or a log scan, you can use the Amazon ECS console to create a CloudWatch Events rule that runs one or more tasks in your cluster at the specified times. Your scheduled event rule can be set to either a specific interval (run every \( N \) minutes, hours, or days), or for more complicated scheduling, you can use a cron expression. For more information, see Schedule Expressions for Rules in the Amazon CloudWatch Events User Guide.

**Note**

This feature is not yet available for Fargate tasks in the following Regions:

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
</tr>
<tr>
<td>EU (London)</td>
<td>eu-west-2</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
</tr>
</tbody>
</table>

Creating a scheduled task

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster in which to create your scheduled task.
4. For Schedule rule name, enter a unique name for your schedule rule. Up to 64 letters, numbers, periods, hyphens, and underscores are allowed.
5. (Optional) For Schedule rule description, enter a description for your rule. Up to 512 characters are allowed.
6. For Schedule rule type, choose whether to use a fixed interval schedule or a cron expression for your schedule rule. For more information, see Schedule Expressions for Rules in the Amazon CloudWatch Events User Guide.
   - For Run at fixed interval, enter the interval and unit for your schedule.
   - For Cron expression, enter the cron expression for your task schedule. These expressions have six required fields, and fields are separated by white space. For more information, and examples of cron expressions, see Cron Expressions in the Amazon CloudWatch Events User Guide.
7. Create a target for your schedule rule.
   - For Target id, enter a unique identifier for your target. Up to 64 letters, numbers, periods, hyphens, and underscores are allowed.
   - For Launch type, choose the launch type for the tasks in your service. For more information, see Amazon ECS Launch Types (p. 227).
c. For **Task definition**, choose the family and revision (family:revision) of the task definition to run for this target.

d. For **Platform version**, choose the platform version to use for this target. For more information, see [AWS Fargate Platform Versions](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/platform-versions.html) (p. 36).

   **Note**
   Platform versions are only applicable to tasks that use the Fargate launch type.

e. For **Number of tasks**, enter the number of instantiations of the specified task definition to run on your cluster when the rule executes.

f. (Optional) For **Task role override**, choose the IAM role to use for the task in your target, instead of the task definition default. For more information, see [IAM Roles for Tasks](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-role-override.html) (p. 435). Only roles with the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For more information about creating an IAM role for your tasks, see [Creating an IAM Role and Policy for your Tasks](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-execution-role.html) (p. 437). You must add `iam:PassRole` permissions for any task role overrides to the CloudWatch IAM role. For more information, see [CloudWatch Events IAM Role](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cloudwatch-events-iam-role.html) (p. 433).

g. If your scheduled task's task definition uses the `awsvpc` network mode, you must configure a VPC, subnet, and security group settings for your scheduled task. For more information, see [Task Networking with the `awsvpc` Network Mode](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-definition-networking.html) (p. 242).

   i. For **Cluster VPC**, if you selected the EC2 launch type, choose the VPC in which your container instances reside. If you selected the Fargate launch type, select the VPC that the Fargate tasks should use. Ensure that the VPC you choose is not configured to require dedicated hardware tenancy as that is not supported by Fargate tasks.

   ii. For **Subnets**, choose the available subnets for your scheduled task placement.

      **Important**
      Only private subnets are supported for the `awsvpc` network mode. Because tasks do not receive public IP addresses, a NAT gateway is required for outbound internet access, and inbound internet traffic should be routed through a load balancer.

   iii. For **Security groups**, a security group has been created for your scheduled tasks, which allows HTTP traffic from the internet (0.0.0.0/0). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.

   iv. For **Auto-assign Public IP**, choose whether to have your tasks receive a public IP address. If you are using Fargate tasks, a public IP address must 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. This allows the task to pull container images.

h. For **CloudWatch Events IAM role for this target**, choose an existing CloudWatch Events service role (`ecsEventsRole`) that you may have already created. Or, choose **Create new role** to create the required IAM role that allows CloudWatch Events to make calls to Amazon ECS to run tasks on your behalf. For more information, see [CloudWatch Events IAM Role](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cloudwatch-events-iam-role.html) (p. 433).

   **Important**
   If your scheduled tasks require the use of the task execution role, or if they use a task role override, then you must add `iam:PassRole` permissions for your task execution role or task role override to the CloudWatch IAM role. For more information, see [CloudWatch Events IAM Role](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/cloudwatch-events-iam-role.html) (p. 433).

   i. (Optional) In the **Container overrides** section, you can expand individual containers and override the command and/or environment variables for that container that are defined in the task definition.

8. (Optional) To add additional targets (other tasks to run when this rule is executed), choose **Add targets** and repeat the previous substeps for each additional target.

9. Choose **Create**.
To edit a scheduled task

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster in which to edit your scheduled task.
4. Select the box to the left of the schedule rule to edit, and choose Edit.
5. Edit the fields to update and choose Update.

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.

**ACTIVATING**

Amazon ECS has to perform additional steps after the task is launched but before the task can transition to the `RUNNING` state. For example, for tasks that have service discovery configured, the service discovery resources must be created.
RUNNING

The task is successfully running.

DEACTIVATING

Amazon ECS has to perform additional steps before the task is stopped.

STOPPING

This is a transition state where Amazon ECS is waiting on the container agent to take further action.

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.

Task Retirement

A task can be scheduled for retirement in the following scenarios:

- Your Fargate tasks are running on a platform version that has a security vulnerability that requires you to launch new tasks using a patched platform version.
- AWS detects the irreparable failure of the underlying hardware hosting the task.

When a task reaches its scheduled retirement date, it is stopped or terminated by AWS. If the task is part of a service, then the task is automatically stopped and the service scheduler starts a new one to replace it. If you are using standalone tasks, then you receive notification of the task retirement and must launch new tasks to replace them.

Identifying Tasks Scheduled for Retirement

If your task is scheduled for retirement, you receive an email before the event with the task ID and retirement date. This email is sent to the address that's associated with your account, the same email address that you use to log in to the AWS Management Console. If you use an email account that you do not check regularly, then you can use the AWS Personal Health Dashboard to determine if any of your tasks are scheduled for retirement. To update the contact information for your account, go to the Account Settings page.

Working with Tasks Scheduled for Retirement

If the task is part of a service, then the task is automatically stopped, The service scheduler starts a new one to replace it after it reaches its scheduled retirement date. If you would like to update your service tasks before the retirement date, you can use the following steps. For more information, see Updating a Service (p. 350).

To update a running service (AWS Management Console)

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the Region that your cluster is in.
3. In the navigation pane, choose Clusters.
4. On the Clusters page, select the name of the cluster in which your service resides.
On the **Cluster**: *name* page, choose **Services**.

6. Check the box to the left of the service to update and choose **Update**.

7. On the **Configure service** page, your service information is pre-populated. Select **Force new deployment** and choose **Next step**.

   **Note**
   For tasks using the Fargate launch type, forcing a new deployment launches new tasks using the patched platform version. Your tasks do not require you select a different platform version. For more information, see AWS Fargate Platform Versions (p. 36).

8. On the **Configure network** and **Set Auto Scaling (optional)** pages, choose **Next step**.

9. Choose **Update Service** to finish and update your service.

### To update a running service (AWS CLI)

1. Obtain the ARN for the service.

   ```bash
   aws ecs list-services --cluster cluster_name --region region
   ```

   **Output:**

   ```json
   {
     "serviceArns": [
       "arn:aws:ecs:region:aws_account_id:service/MyService"
     ]
   }
   ```

2. Update your service, forcing a new deployment that deploys new tasks.

   ```bash
   aws ecs update-service --service serviceArn --force-new-deployment --cluster cluster_name --region region
   ```

   If you are using standalone tasks, then you can start a new task to replace it. For more information, see Running Tasks (p. 269).

### Fargate Task Recycling

A task can be recycled in the following scenarios:

- The task is using the Fargate launch type and using platform version 1.3.0 or later. For more information, see AWS Fargate Platform Versions (p. 36).
- The task is part of an Amazon ECS service. Standalone tasks are not affected by task recycling, but may still be scheduled for retirement. For more information, see Task Retirement (p. 287).
- AWS determines there is cause for the task to be recycled, as described below.

When AWS determines that a security or infrastructure update is needed for a Fargate task, it will apply the necessary patches for the task. Most of these patches will be transparent and the task will not need to be stopped, but on occasion it is necessary for the task to be recycled. Starting with Fargate platform version 1.3.0, any Fargate tasks launched as part of a service may be stopped and a new one started by the Amazon ECS service scheduler in order to provide the best possible security and availability for the task. Task recycling begin after February 1, 2019 and will continue on a rolling basis. The service scheduler will ensure that the desired task count for your service will be maintained.
To prepare for this new process, we recommend testing your application behavior by simulating this scenario. You can do this by stopping an individual task in your service to test for resiliency.

Creating a Scheduled Task Using the AWS CLI

This topic shows you how to create a scheduled task using the AWS CLI. The scheduled task creation uses the CloudWatch Events API. For more information, see What is Amazon CloudWatch Events? in the Amazon CloudWatch Events User Guide.

Complete the following prerequisites:

- Set up an AWS account.
- Install and configure the AWS CLI. For more information, see AWS Command Line Interface.

To create the scheduled task

1. Create the CloudWatch Events rule. This example creates a rule named `MyRule1` that is triggered every day at 12:00pm UTC.

   ```
   aws events put-rule --schedule-expression "cron(0 12 * * ? *)" --name MyRule1
   ```

   **Note**
   For other examples of rule expressions, see Schedule Expressions for Rules in the Amazon CloudWatch Events User Guide.

2. Add the details of your ECS cluster and task definition as a target for the CloudWatch Events rule. Specify the cluster and task definition using the full ARN.

   This example defines the target for `MyRule1` as the `first-run-task-definition:1` task definition in the `default` cluster and assigns the `ecsEventsRole` IAM role to it. It requests that 1 task be scheduled. The cluster and task definition must already be created. Otherwise, you receive an error.

   ```
   aws events put-targets --rule "MyRule1" --targets "Id"="1","Arn"="arn:aws:ecs:us-east-1:123456789012:cluster/default","RoleArn"="arn:aws:iam::123456789012:role/ecsEventsRole","EcsParameters"="{"TaskDefinitionArn"="arn:aws:ecs:us-east-1:123456789012:task-definition/first-run-task-definition:1","TaskCount"=1}"
   ```
Services

Amazon ECS allows you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. This is called a service. 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 and maintain the desired count of tasks in the service depending on the scheduling strategy used.

In addition to maintaining the desired count of tasks in your service, you can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

Topics
- Service Scheduler Concepts (p. 290)
- Additional Service Concepts (p. 292)
- Service Definition Parameters (p. 292)
- Amazon ECS Deployment Types (p. 299)
- Service Load Balancing (p. 306)
- Service Auto Scaling (p. 321)
- Service Discovery (p. 328)
- Creating a Service (p. 340)
- Updating a Service (p. 350)
- Deleting a Service (p. 353)
- Service Throttle Logic (p. 354)

Service Scheduler Concepts

If a task in a service stops, the task is killed and a new task is launched. This process continues until your service reaches the number of desired running tasks based on the scheduling strategy that you specified.

The service scheduler includes logic that throttles how often tasks are restarted if they repeatedly fail to start. If a task is stopped without having entered a RUNNING state, determined by the task having a startedAt time stamp, the service scheduler starts to incrementally slow down the launch attempts and emits a service event message. This behavior prevents unnecessary resources from being used for failed tasks, giving you a chance to resolve the issue. After the service is updated, the service scheduler resumes normal behavior. For more information, see Service Throttle Logic (p. 354) and Service Event Messages (p. 591).

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 (p. 291).
- 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 (p. 291).
**Note**
Fargate tasks do not support the DAEMON scheduling strategy.

### Daemon

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. 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.

The daemon service scheduler does not place any tasks on instances that have the DRAINING status. If a container instance transitions to DRAINING, the daemon tasks on it are stopped. It also monitors when new container instances are added to your cluster and adds the daemon tasks to them.

If deploymentConfiguration is specified, the maximum percent parameter must be 100. The default value for a daemon service for maximumPercent is 100%. The default value for a daemon service for minimumHealthyPercent is 0% for the AWS CLI, the AWS SDKs, and the APIs, and 50% for the AWS Management Console.

**Note**
The daemon service scheduler does not support the use of Classic Load Balancers.

### 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.

When the service scheduler, using the REPLICA strategy, launches new tasks or stops running tasks that use the Fargate launch type, it attempts to maintain balance across the Availability Zones in your service.

When the service scheduler, using the REPLICA strategy, launches new tasks using the EC2 launch type, the scheduler uses 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).
- Determine which container instances satisfy any placement constraints that are defined for the service.
- If there is a placement strategy defined, use that strategy to select an instance from the remaining candidates.
- If there is no placement strategy defined, balance tasks across the Availability Zones in your cluster with the following logic:
  - Sort the valid container instances, giving 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.
  - 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, using the REPLICA strategy, stops running tasks, it attempts to maintain balance across the Availability Zones in your cluster. For tasks using the EC2 launch type, 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, then a task is selected that leaves the remaining tasks with the best spread.
• If no placement strategy is defined, maintain balance across the Availability Zones in your cluster with the following logic:
  • Sort the valid container instances, giving 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, 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.

Additional Service Concepts

• You can optionally run your service behind a load balancer. For more information, see Service Load Balancing (p. 306).

• You can optionally specify a deployment configuration for your service. During a deployment (which is triggered by updating the task definition or desired count of a service), the service scheduler uses the minimum healthy percent and maximum percent parameters to determine the deployment strategy. For more information, see Service Definition Parameters (p. 292).

• You can optionally configure your service to use Amazon ECS service discovery. Service discovery uses Amazon Route 53 auto naming APIs to manage DNS entries for your service's tasks, making them discoverable within your VPC. For more information, see Service Discovery (p. 328).

Service Definition Parameters

A service definition defines which task definition to use with your service, how many instantiations of that task to run, which load balancers (if any) to associate with your tasks, as well as other service parameters.

```json
{
   "cluster": "",
   "serviceName": "",
   "taskDefinition": "",
   "loadBalancers": [
   {
      "targetGroupArn": "",
      "loadBalancerName": "",
      "containerName": "",
      "containerPort": 0
   }
   ],
   "serviceRegistries": [
   {
      "registryArn": "",
      "port": 0,
      "containerName": "",
      "containerPort": 0
   }
   ],
   "desiredCount": 0,
   "clientToken": "",
   "launchType": "EC2",
   "platformVersion": "",
   "role": "",
   "deploymentConfiguration": {
      "maximumPercent": 0,
      "minimumHealthyPercent": 0
   }
}
```
Note
You can create the above service definition template with the following AWS CLI command.

```
aws ecs create-service --generate-cli-skeleton
```

You can specify the following parameters in a service definition.

**cluster**
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.

**serviceName**
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.

Required: Yes

**taskDefinition**
The family and revision (family:revision) or full Amazon Resource Name (ARN) of the task definition to run in your service. If a revision is not specified, the latest ACTIVE revision is used.
loadBalancers

A load balancer object representing the load balancer to use with your service. Currently, you are limited to one load balancer or target group per service. After you create a service, the load balancer name or target group ARN, container name, and container port specified in the service definition are immutable.

For Classic Load Balancers, this object must contain the load balancer name, 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 is registered with the load balancer specified here.

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 here.

targetGroupArn

The full Amazon Resource Name (ARN) of the Elastic Load Balancing target group associated with a service.

loadBalancerName

The name of the load balancer.

containerName

The name of the container (as it appears in a container definition) to associate with the load balancer.

containerPort

The port on the container to associate with the load balancer. This port must correspond to a containerPort in the service's task definition. Your container instances must allow ingress traffic on the hostPort of the port mapping.

serviceRegistries

The details of the service discovery configuration for your service. For more information, see Service Discovery (p. 328).

registryArn

The Amazon Resource Name (ARN) of the service registry. The currently supported service registry is Amazon Route 53 Auto Naming. For more information, see Service.

port

The port value 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

The container name value, already specified in the task definition, to be used for your service discovery service. 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

The port value, already specified in the task definition, to be used for your service discovery service. 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.
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.

desiredCount

The number of instantiations of the specified task definition to place and keep running on your cluster.

clientToken

Unique, case-sensitive identifier you provide to ensure the idempotency of the request. Up to 32 ASCII characters are allowed.

launchType

The launch type on which to run your service. Accepted values are FARGATE or EC2. If a launch type is not specified, EC2 is used by default. For more information, see Amazon ECS Launch Types (p. 227).

platformVersion

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.

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 AWS Fargate Platform Versions (p. 36).

**Note**
Platform versions are not specified for tasks using the EC2 launch type.

role

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 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.

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 (p. 421).

deploymentConfiguration

Optional deployment parameters that control how many tasks run during the deployment and the ordering of stopping and starting tasks.

maximumPercent

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 or PENDING state during a deployment, as a percentage of the desiredCount (rounded down to the nearest integer). This parameter enables you 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 may start four new tasks before stopping the four older tasks (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%.

If your service is using the DAEMON service scheduler type, the maximumPercent should remain at 100%, which 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 maximumPercent value is set to the default value and 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. If the tasks in the service use the Fargate launch type, the maximum percent value is not used, although it is returned when describing your service.

minimumHealthyPercent

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, as a percentage of the desiredCount (rounded up to the nearest integer). This parameter enables you to deploy without using additional cluster capacity. For example, if your service has a desiredCount of four tasks and a minimumHealthyPercent of 50%, the service scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that do not use a load balancer are considered healthy if they are in the RUNNING state. Tasks for services that do use a load balancer are considered healthy if they are in the RUNNING state and the container instance on which the load balancer is hosted is reported as healthy. The default value for a replica service for minimumHealthyPercent is 50% in the AWS Management Console and 100% for the AWS CLI, the AWS SDKs, and the APIs. 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 tasks that use the EC2 launch type, the minimum healthy percent value is set to the default value and 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. If the tasks in the service use the Fargate launch type, the minimum healthy percent value is not used, although it is returned when describing your service.

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 are not supported.

type

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.

type

expression

A cluster query language expression to apply to the constraint. Note you cannot specify an expression if the constraint type is distinctInstance. For more information, see Cluster Query Language (p. 280).
placementStrategy

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

type

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 is 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

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 is 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.

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 is not supported for other network modes. If using the Fargate launch type, the awsvpc network mode is required. For more information, see Task Networking with the awsvpc Network Mode (p. 242).

awsvpcConfiguration

An object representing the subnets and security groups for a task or service.

subnets

The subnets associated with the task or service.

securityGroups

The security groups associated with the task or service. If you do not specify a security group, the default security group for the VPC is used.

assignPublicIP

Whether the task's elastic network interface receives a public IP address.

healthCheckGracePeriodSeconds

The period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks, container health checks, and Route 53 health checks after a task has first started. This is only valid if your service is configured to use a load balancer. 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 during which the ECS service scheduler ignores the health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

schedulingStrategy

The scheduling strategy to use. For more information, see Service Scheduler Concepts (p. 290).

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 (p. 291).
• **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 (p. 291).

  **Note**
  Fargate tasks do not support the DAEMON scheduling strategy.

**deploymentController**

The deployment controller to use for the service. For more information, see Amazon ECS Deployment Types (p. 299).

**type**

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**
  
  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.

**tags**

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. Tag keys can have a maximum character length of 128 characters, and tag values can have a maximum length of 256 characters. For more information, see Tagging Your Amazon ECS Resources (p. 359).

- **key**
  
  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**
  
  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 enable Amazon ECS managed tags for the tasks in the service. For more information, see Tagging Your Resources for Billing (p. 361).

**propagateTags**

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, use the TagResource API action.
Amazon ECS Deployment Types

An Amazon ECS deployment type determines the deployment strategy that your service uses. There are three deployment types: rolling update, blue/green, and external.

Topics
- Rolling Update (p. 299)
- Blue/Green Deployment with CodeDeploy (p. 299)
- External Deployment (p. 302)

Rolling Update

The rolling update deployment type is controlled by Amazon ECS. This involves the service scheduler replacing the current running version of the container with the latest version. The number of tasks that Amazon ECS adds or removes from the service during a rolling update is controlled by the deployment configuration. A deployment configuration consists of the minimum and maximum number of tasks allowed during a service deployment.

To create a new Amazon ECS service that uses the rolling update deployment type, see Creating a Service (p. 340).

Blue/Green Deployment with CodeDeploy

The blue/green deployment type uses the blue/green deployment model controlled by CodeDeploy. This deployment type enables you 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.

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 only supported value at this time is CodeDeployDefault.AllAtOnce, which means all traffic is routed from the original task set to the replacement task set at the same time.

For more information, see Working with Deployment Configurations in the AWS CodeDeploy User Guide.
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.

Blue/Green Deployment 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. Classic Load Balancers aren't supported. 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.
- Service auto scaling is not supported when using the blue/green deployment type.
- When you initially create an 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 an 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.

Amazon ECS Console Experience

The service create and service update workflows in the Amazon ECS console supports blue/green deployments.

To create an Amazon ECS service that uses the blue/green deployment type, see Creating a Service (p. 340).

To update an existing Amazon ECS service that is using the blue/green deployment type, see Updating a Service (p. 350).

When you use the Amazon ECS console to create an Amazon ECS service using the blue/green deployment type, an Amazon ECS task set and the following CodeDeploy resources are created automatically with the following default settings.
Resource | Default Setting
--- | ---
Application name | AppECS-<cluster[:47]>-<service[:47]>
Deployment group name | DgpECS-<cluster[:47]>-<service[:47]>
Deployment group load balancer info | The load balancer production listener, optional test listener, and target groups specified are added to the deployment group configuration.
Traffic rerouting settings | Traffic rerouting – The default setting is **Reroute traffic immediately**. You can change it on the CodeDeploy console or by updating the TrafficRoutingConfig. For more information, see CreateDeploymentConfig in the AWS CodeDeploy API Reference.
Original revision termination settings | The original revision termination settings are configured to wait 1 hour after traffic has been rerouted before terminating the blue task set.
Deployment configuration | The deployment configuration is set to CodeDeployDefault.AllAtOnce, which routes all traffic at one time from the blue task set to the green task set. You can't change this setting.
Automatic rollback configuration | If a deployment fails, the automatic rollback settings are configured to roll it back.

To view details of an Amazon ECS service using the blue/green deployment type, use the **Deployments** tab on the Amazon ECS console.

To view the details of a CodeDeploy deployment group in the CodeDeploy console, see **View Deployment Group Details with CodeDeploy** in the AWS CodeDeploy User Guide.

To modify the settings for an CodeDeploy deployment group in the CodeDeploy console, see **Change Deployment Group Settings with CodeDeploy** in the AWS CodeDeploy User Guide.

**Blue/Green Deployment Required IAM Permissions**

Amazon ECS blue/green deployments are made possible by a combination of the Amazon ECS and CodeDeploy APIs. IAM 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 (p. 408).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy:CreateApplication",
        "codedeploy:CreateDeployment",
        "codedeploy:CreateDeploymentGroup",
        "codedeploy:GetApplication",
        "codedeploy:GetDeployment",
```

---

API Version 2014-11-13

301
Note

In addition to the standard Amazon ECS permissions required to run tasks and services, IAM 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 (p. 428).

The Create Services (p. 448) and Update Services (p. 448) IAM policy examples show the permissions that are required for IAM users to use Amazon ECS blue/green deployments on the AWS Management Console.

External Deployment

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. 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 `DeleteTask`). Each API action has a subset of the service definition parameters that it can manage.

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.

**Important**

Service auto scaling is not supported when using an external deployment controller.

The following is the basic workflow to 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**
   
   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.

   Required: Yes

   **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. For more information, see deploymentConfiguration.

   **tags**
   
   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. Tag keys can have a maximum character length of 128 characters, and tag values can have a maximum length of 256 characters. For more information, see Tagging Your Amazon ECS Resources (p. 359).

   **key**
   
   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**
   
   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 enable Amazon ECS managed tags for the tasks within the service. For more information, see Tagging Your Resources for Billing (p. 361).

   **propagateTags**
   
   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, use the TagResource API action.

   **healthCheckGracePeriodSeconds**
   
   The period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks, container health checks, and Route 53 health
checks after a task has first started. This is only valid if your service is configured to use a load balancer. 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 during which the ECS service scheduler ignores the health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

**schedulingStrategy**

The scheduling strategy to use. Services using an external deployment controller support only the REPLICA scheduling strategy. For more information, see Service Scheduler Concepts (p. 290).

**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 a service using an external deployment controller.

```json
{
  "cluster": "",
  "serviceName": "",
  "desiredCount": 0,
  "role": "",
  "deploymentConfiguration": {
    "maximumPercent": 0,
    "minimumHealthyPercent": 0
  },
  "placementConstraints": [
    {
      "type": "distinctInstance",
      "expression": ""
    }
  ],
  "placementStrategy": [
    {
      "type": "binpack",
      "field": ""
    }
  ],
  "healthCheckGracePeriodSeconds": 0,
  "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**
   
   The launch type on which to run your service. Accepted values are FARGATE or EC2. If a launch type is not specified, EC2 is used by default. For more information, see Amazon ECS Launch Types (p. 227).

   **platformVersion**
   
   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.

   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 AWS Fargate Platform Versions (p. 36).

   **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.

   **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, see Task Networking with the awsvpc Network Mode (p. 242).

   **serviceRegistries**
   
   The details of the service discovery registries to assign to this service. For more information, see Service Discovery (p. 328).

   **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.

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.
Service Load Balancing

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

Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers, and Amazon ECS services can use either type of load balancer. Application Load Balancers are used to route HTTP/HTTPS (or Layer 7) traffic. Network Load Balancers and Classic Load Balancers are used to route TCP (or Layer 4) traffic. For more information, see Load Balancer Types (p. 307).

Application Load Balancers offer several features that make them attractive for use with Amazon ECS services:

- 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).

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 Classic Load Balancers. For more information about Elastic Load Balancing and the differences between the load balancer types, see the Elastic Load Balancing User Guide.

Topics
- Load Balancing Concepts (p. 306)
- Load Balancer Types (p. 307)
- Check the Service Role for Your Account (p. 310)
- Creating a Load Balancer (p. 311)

Load Balancing Concepts

- All of the containers that are launched in a single task definition are always placed on the same container instance. For Classic Load Balancers, you may choose to put multiple containers (in the same task definition) behind the same load balancer by defining multiple host ports in the service definition and adding those listener ports to the load balancer. For example, if a task definition consists of Elasticsearch using port 3030 on the container instance, with Logstash and Kibana using port 4040 on the container instance, the same load balancer can route traffic to Elasticsearch and Kibana through two listeners. For more information, see Listeners for Your Classic Load Balancer in the User Guide for Classic Load Balancers.

Important
We do not recommend connecting multiple services to the same Classic Load Balancer. Because entire container instances are registered and deregistered with Classic Load Balancers (and not host and port combinations), this configuration can cause issues if a task from one service stops, causing the entire container instance to be deregistered from the Classic Load Balancer while another task from a different service on the same container instance is still using it. If you want to connect multiple services to a single load balancer (for example, to save costs), we recommend using an Application Load Balancer.

- There is a limit of one load balancer or target group per service.
- Services with tasks that use the awsvpc network mode (for example, those with the Fargate launch type) only support Application Load Balancers and Network Load Balancers. Classic Load Balancers are not supported. Also, 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 elastic network interface, not an Amazon EC2 instance.

- If your service using an Application Load Balancer requires access to multiple load balanced ports (for example, port 80 and port 443 for an HTTP/HTTPS service), you can configure two listeners: one listener responsible for HTTPS that will forward the request to the service, and another listener that will be 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.

- Container health checks are not supported for tasks that are part of a service that is configured to use a Classic Load Balancer.

- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.

- After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

- If a service's task fails the load balancer health check criteria, the task is killed and restarted. This process continues until your service reaches the number of desired running tasks.

- If you configure your Application Load Balancer to use slow start mode, you must configure your task health check to return an UNHEALTHY status until after the slow start period is over. For more information about slow start mode, see Target Groups for Your Application Load Balancers.

- If you are experiencing problems with your load balancer-enabled services, see Troubleshooting Service Load Balancers (p. 597).

### Load Balancer Types

Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. Amazon ECS services can use either type of load balancer. Application Load Balancers are used to route HTTP/HTTPS (or Layer 7) traffic. Network Load Balancers and Classic Load Balancers are used to route TCP (or Layer 4) traffic.

**Topics**

- Application Load Balancer (p. 307)
- Network Load Balancer (p. 308)
- Classic Load Balancer (p. 309)

### Application Load Balancer

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 is launched, 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.
Network Load Balancer

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.
Classic Load Balancer

A Classic Load Balancer makes routing decisions at either the transport layer (TCP/SSL) or the application layer (HTTP/HTTPS). Classic Load Balancers currently require a fixed relationship between the load balancer port and the container instance port. For example, it is possible to map the load balancer port 80 to the container instance port 3030 and the load balancer port 4040 to the container instance port 4040. However, it is not possible to map the load balancer port 80 to port 3030 on one container instance and port 4040 on another container instance. This static mapping requires that your cluster has at least as many container instances as the desired count of a single service that uses a Classic Load Balancer. For more information, see the User Guide for Classic Load Balancers.
Check the Service Role for Your Account

Amazon ECS needs permissions to register and deregister container instances with your load balancer when tasks are created and stopped.

In most cases, the Amazon ECS service role is automatically created for you in the Amazon ECS console first run experience. You can use the following procedure to check and see if your account already has an Amazon ECS service role.

To check for the **ecsServiceRole** in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for **ecsServiceRole**. If the role does not exist, see Amazon ECS Service Scheduler IAM Role (p. 426) to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**.
5. In the **Managed Policies** section, ensure that the **AmazonEC2ContainerServiceRole** managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose **Attach Policy**.
   b. For **Filter**, type **AmazonEC2ContainerServiceRole** to narrow the available policies to attach.
   c. Select the box to the left of the **AmazonEC2ContainerServiceRole** policy and choose **Attach Policy**.
6. Choose **Trust Relationships, Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose Cancel. If the trust relationship does not match, copy the policy into the Policy Document window and choose Update Trust Policy.

```json
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Creating a Load Balancer

This section provides a hands-on introduction to using Elastic Load Balancing through the AWS Management Console to use with your Amazon ECS services. In this section, you create an external load balancer that receives public network traffic and routes it to your Amazon ECS container instances.

Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers, and Amazon ECS services can use either type of load balancer. Application Load Balancers are used to route HTTP/HTTPS traffic. Network Load Balancers and Classic Load Balancers are used to route TCP or Layer 4 traffic.

Application Load Balancers offer several features that make them attractive for use with Amazon ECS services:

- 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).

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of these latest features. For more information about Elastic Load Balancing and the differences between the load balancer types, see the Elastic Load Balancing User Guide.

**Note**
Currently, Amazon ECS services can only specify a single load balancer or target group. If your service requires access to multiple load balanced ports (for example, port 80 and port 443 for an HTTP/HTTPS service), you must use a Classic Load Balancer with multiple listeners. To use an Application Load Balancer, separate the single HTTP/HTTPS service into two services, where each handles requests for different ports. Then, each service could use a different target group behind a single Application Load Balancer.

**Topics**
- Creating an Application Load Balancer (p. 312)
- Creating a Network Load Balancer (p. 316)
- Creating a Classic Load Balancer (p. 317)
Creating an Application Load Balancer

This section walks you through the process of creating an Application Load Balancer in the AWS Management Console.

Define Your Load Balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A listener is a process that checks for connection requests. It is configured with a protocol and a port for the frontend (client to load balancer) connections, and protocol and a port for the backend (load balancer to backend instance) connections. In this example, you configure a listener that accepts HTTP requests on port 80 and sends them to the containers in your tasks on port 80 using HTTP.

To define your load balancer

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under LOAD BALANCING, choose Load Balancers.
4. Choose Create Load Balancer.
5. On the Select load balancer type page, choose Application Load Balancer and then choose Continue.
6. Complete the Configure Load Balancer page as follows:
   a. For Name, type a name for your load balancer.
   b. For Scheme, an internet-facing load balancer routes requests from clients over the internet to targets. An internal load balancer routes requests to targets using private IP addresses.
   c. For IP address type, choose ipv4 to support IPv4 addresses only or dualstack to support both IPv4 and IPv6 addresses.
   d. For Listeners, the default is a listener that accepts HTTP traffic on port 80. You can keep the default listener settings, modify the protocol or port of the listener, or choose Add to add another listener.
      
      **Note**
      
      If you plan on routing traffic to more than one target group, see ListenerRules for details on how to add host or path-based rules.
   e. For VPC, select the same VPC that you used for the container instances on which you intend to run your service.
   f. For Availability Zones, select the check box for the Availability Zones to enable for your load balancer. If there is one subnet for that Availability Zone, it is selected. If there is more than one subnet for that Availability Zone, select one of the subnets. You can select only one subnet per Availability Zone. Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
   g. Choose Next: Configure Security Settings.

(Optional) Configure Security Settings

If you created a secure listener in the previous step, complete the Configure Security Settings page as follows; otherwise, choose Next: Configure Security Groups.
To configure security settings

1. If you have a certificate from AWS Certificate Manager, choose **Choose an existing certificate from AWS Certificate Manager (ACM)**, and then choose the certificate from **Certificate name**.
2. If you have already uploaded a certificate using IAM, choose **Choose an existing certificate from AWS Identity and Access Management (IAM)**, and then choose your certificate from **Certificate name**.
3. If you have a certificate ready to upload, choose **Upload a new SSL Certificate to AWS Identity and Access Management (IAM)**. For **Certificate name**, type a name for the certificate. For **Private Key**, copy and paste the contents of the private key file (PEM-encoded). In **Public Key Certificate**, copy and paste the contents of the public key certificate file (PEM-encoded). In **Certificate Chain**, copy and paste the contents of the certificate chain file (PEM-encoded), unless you are using a self-signed certificate and it’s not important that browsers implicitly accept the certificate.
4. For **Select policy**, choose a predefined security policy. For details on the security policies, see **Security Policies**.
5. Choose **Next: Configure Security Groups**.

Configure Security Groups

You must assign a security group to your load balancer that allows inbound traffic to the ports that you specified for your listeners. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To assign a security group to your load balancer

1. On the **Assign Security Groups** page, choose **Create a new security group**.
2. Enter a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your listener to use.

**Note**

Later in this topic, you create a security group rule for your container instances that allows traffic on all ports coming from the security group created here, so that the Application Load Balancer can route traffic to dynamically assigned host ports on your container instances.
Creating a Load Balancer

3. Choose Next: Configure Routing to go to the next page in the wizard.

Configure Routing

In this section, you create a target group for your load balancer and the health check criteria for targets that are registered within that group.

To create a target group and configure health checks

1. For Target group, keep the default, New target group.
2. For Name, type a name for the new target group.
3. Set Protocol and Port as needed.
4. For Target type, choose whether to register your targets with an instance ID or an IP address.
   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.
5. For Health checks, keep the default health check settings.
6. Choose Next: Register Targets.

Register Targets

Your load balancer distributes traffic between the targets that are registered to its target groups. When you associate a target group to an Amazon ECS service, Amazon ECS automatically registers and deregisters containers with your target group. Because Amazon ECS handles target registration, you do not add targets to your target group at this time.
To skip target registration

1. In the Registered instances section, ensure that no instances are selected for registration.
2. Choose Next: Review to go to the next page in the wizard.

Review and Create

Review your load balancer and target group configuration and choose Create to create your load balancer.

Create a Security Group Rule for Your Container Instances

After your Application Load Balancer has been created, you must add an inbound rule to your container instance security group that allows traffic from your load balancer to reach the containers.

To allow inbound traffic from your load balancer to your container instances

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the left navigation, choose Security Groups.
3. Choose the security group that your container instances use. If you created your container instances by using the Amazon ECS first run wizard, this security group may have the description, ECS Allowed Ports.
4. Choose the Inbound tab, and then choose Edit.
5. For Type, choose All traffic.
6. For Source, choose Custom, and then type the name of your Application Load Balancer security group that you created in Configure Security Groups (p. 313). This rule allows all traffic from your Application Load Balancer to reach the containers in your tasks that are registered with your load balancer.
7. Choose Save to finish.
Create an Amazon ECS Service

After your load balancer and target group are created, you can specify the target group in a service definition when you create a service. When each task for your service is started, the container and port combination specified in the service definition is registered with your target group and traffic is routed from the load balancer to that container. For more information, see Creating a Service (p. 340).

Creating a Network Load Balancer

This section walks you through the process of creating a Network Load Balancer in the AWS Management Console.

Define Your Load Balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A listener is a process that checks for connection requests. It is configured with a protocol and port for the frontend (client to load balancer) connections, and a protocol and port for the backend (load balancer to backend instance) connections. In this example, you configure an Internet-facing load balancer in the selected network with a listener that receives TCP traffic on port 80.

To define your load balancer

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under LOAD BALANCING, choose Load Balancers.
4. Choose Create Load Balancer.
5. On the Select load balancer type page, choose Create under Network Load Balancer.
6. Complete the Configure Load Balancer page as follows:
   a. For Name, type a name for your load balancer.
   b. For Scheme, choose either internet-facing or internal. An internet-facing load balancer routes requests from clients over the internet to targets. An internal load balancer routes requests to targets using private IP addresses.
   c. For Listeners, the default is a listener that accepts TCP traffic on port 80. You can keep the default listener settings, modify the protocol or port of the listener, or choose Add listener to add another listener.
      
      Note
      If you plan on routing traffic to more than one target group, see ListenerRules for details on how to add host or path-based rules.
   d. For Availability Zones, select the VPC that you used for your Amazon EC2 instances. For each Availability Zone that you used to launch your Amazon EC2 instances, select an Availability Zone and then select the public subnet for that Availability Zone. To associate an Elastic IP address with the subnet, select it from Elastic IP.
   e. Choose Next: Configure Routing.

Configure Routing

You register targets, such as Amazon EC2 instances, with a target group. The target group that you configure in this step is used as the target group in the listener rule, which forwards requests to the target group. For more information, see Target Groups for Your Network Load Balancers.
To configure your target group

1. For **Target group**, keep the default, **New target group**.
2. For **Name**, type a name for the target group.
3. Set **Protocol** and **Port** as needed.
4. For **Target type**, choose whether to register your targets with an instance ID or an IP address.
   
   **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.
5. For **Health checks**, keep the default health check settings.
6. Choose **Next: Register Targets**.

Register Targets with the Target Group

Your load balancer distributes traffic between the targets that are registered to its target groups. When you associate a target group to an Amazon ECS service, Amazon ECS automatically registers and deregisters containers with your target group. Because Amazon ECS handles target registration, you do not add targets to your target group at this time.

To skip target registration

1. In the **Registered instances** section, ensure that no instances are selected for registration.
2. Choose **Next: Review** to go to the next page in the wizard.

Review and Create

Review your load balancer and target group configuration and choose **Create** to create your load balancer.

Create an Amazon ECS Service

After your load balancer and target group are created, you can specify the target group in a service definition when you create a service. When each task for your service is started, the container and port combination specified in the service definition is registered with your target group and traffic is routed from the load balancer to that container. For more information, see **Creating a Service (p. 340)**.

Creating a Classic Load Balancer

This section walks you through the process of creating a Classic Load Balancer in the AWS Management Console.

You can create your Classic Load Balancer for use with EC2-Classic or a VPC. Some of the tasks described in these procedures apply only to load balancers in a VPC.

Define Your Load Balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.
A listener is a process that checks for connection requests. It is configured with a protocol and port for the frontend (client to load balancer) connections and a protocol, and a port for the backend (load balancer to backend instance) connections. In this example, you configure a listener that accepts HTTP requests on port 80 and sends them to the backend instances on port 80 using HTTP.

To define your load balancer

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under LOAD BALANCING, choose Load Balancers.
4. Choose Create Load Balancer.
5. On the Select load balancer type page, choose Classic Load Balancer.
6. For Load Balancer name, enter a unique name for your load balancer.

   The load balancer name you choose must be unique within your set of load balancers, must have a maximum of 32 characters, and must only contain alphanumeric characters or hyphens.
7. For Create LB inside, select the same network that your container instances are located in: EC2-Classic or a specific VPC.
8. The default values configure an HTTP load balancer that forwards traffic from port 80 at the load balancer to port 80 of your container instances, but you can modify these values for your application. For more information, see Listeners for Your Classic Load Balancer in the User Guide for Classic Load Balancers.
9. [EC2-VPC] To improve the availability of your load balancer, select at least two subnets in different Availability Zones. Your load balancer subnet configuration must include all Availability Zones that your container instances reside in. In the Select Subnets section, under Available Subnets, select the subnets. The subnets that you select are moved under Selected Subnets.

   Note
   If you selected EC2-Classic as your network, or you have a default VPC but did not choose Enable advanced VPC configuration, you do not see Select Subnets.

   Available Subnets

<table>
<thead>
<tr>
<th>Actions</th>
<th>Availability Zone</th>
<th>Subnet ID</th>
<th>Subnet CIDR</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>us-west-2c</td>
<td>subnet-cb663da2</td>
<td>10.0.0.24</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>us-west-2c</td>
<td>subnet-c9663da0</td>
<td>10.0.0.24</td>
<td></td>
</tr>
</tbody>
</table>

   Selected Subnets

<table>
<thead>
<tr>
<th>Actions</th>
<th>Availability Zone</th>
<th>Subnet ID</th>
<th>Subnet CIDR</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>us-west-2a</td>
<td>subnet-e4f33493</td>
<td>10.0.2.0/24</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>us-west-2b</td>
<td>subnet-5264e837</td>
<td>10.0.3.0/24</td>
<td></td>
</tr>
</tbody>
</table>

10. Choose Next: Assign Security Groups to go to the next page in the wizard.

Assign a Security Group to Your Load Balancer in a VPC

If you created your load balancer in a VPC, you must assign it a security group that allows inbound traffic to the ports that you specified for your load balancer and the health checks for your load balancer.
Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

**Note**
If you selected EC2-Classic as your network, you do not see this page in the wizard and you can go to the next step. Elastic Load Balancing provides a security group that is assigned to your load balancer for EC2-Classic automatically.

**To assign a security group to your load balancer**

1. On the **Assign Security Groups** page, choose **Create a new security group**.
2. Enter a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your load balancer to use. If you specified a different port for the health checks, you must choose **Add Rule** to add a rule that allows inbound traffic to that port as well.

**Note**
Also assign this security group to container instances in your service, or another security group with the same rules.

3. Choose **Next: Configure Security Settings** to go to the next page in the wizard.

**(Optional) Configure Security Settings**

For this tutorial, you can choose **Next: Configure Health Check** to continue to the next step. For more information about creating an HTTPS load balancer and using additional security features, see HTTPS Load Balancers in the User Guide for Classic Load Balancers.

**Configure Health Checks for Your EC2 Instances**

Elastic Load Balancing automatically checks the health of the tasks in your service. If Elastic Load Balancing finds an unhealthy task, it stops sending traffic to the instance and reroutes traffic to healthy instances. Amazon ECS stops your unhealthy task and starts another instance of that task.

**Note**
The following procedure configures an HTTP (port 80) load balancer, but you can modify these values for your application.

**To configure a health check for your instances**

1. On the **Configure Health Check** page, do the following:
a. Leave **Ping Protocol** set to its default value of **HTTP**.

b. Leave **Ping Port** set to its default value of **80**.

c. For **Ping Path**, replace the default value with a single forward slash ("/"). This tells Elastic Load Balancing to send health check queries to the default home page for your web server, such as `index.html` or `default.html`.

d. Leave the other fields at their default values.

### Configure Health Check

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize these settings to match your specific needs.

<table>
<thead>
<tr>
<th>Ping Protocol</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping Port</td>
<td>80</td>
</tr>
<tr>
<td>Ping Path</td>
<td>/</td>
</tr>
</tbody>
</table>

2. Choose **Next: Add EC2 Instances** to go to the next page in the wizard.

### Load Balancer Instance Registration

Your load balancer distributes traffic between the instances that are registered to it. When you assign your load balancer to an Amazon ECS service, Amazon ECS automatically registers and deregisters container instances when tasks from your service are running on them. Because Amazon ECS handles container instance registration, you do not add container instances to your load balancer at this time.

**To skip instance registration and tag the load balancer**

1. On the **Add EC2 Instances** page, for **Add Instances to Load Balancer**, ensure that no instances are selected for registration.
2. Leave the other fields at their default values.
3. Choose **Next: Add Tags** to go to the next page in the wizard.

### Tag Your Load Balancer (Optional)

You can tag your load balancer, or continue to the next step. You can tag your load balancer later on. For more information, see [Tag Your Classic Load Balancer](#) in the [User Guide for Classic Load Balancers](#).

**To add tags to your load balancer**

1. On the **Add Tags** page, specify a key and a value for the tag.
2. To add another tag, choose **Create Tag** and specify a key and a value for the tag.
3. After you are finished adding tags, choose **Review and Create**.

### Create and Verify Your Load Balancer

Before you create the load balancer, review the settings that you selected. After creating the load balancer, you can create a service that uses it to verify that it's sending traffic to your container instances.
To finish creating your load balancer

1. On the Review page, check your settings. To change the initial settings, choose the corresponding edit link.
2. Choose Create to create your load balancer.
3. After you are notified that your load balancer was created, choose Close.

Create an Amazon ECS Service

After your load balancer is created, you can specify it in a service definition when you create a service. For more information, see Creating a Service (p. 340).

Service Auto Scaling

Your Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. Service Auto Scaling leverages the Application Auto Scaling service to provide this functionality. Service Auto Scaling is available in all regions that support Amazon ECS. For more information, see the Application Auto Scaling User Guide.

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. You can use these service utilization metrics to scale your service out to deal with high demand at peak times, and to scale your service in to reduce costs during periods of low utilization. For more information, see Service Utilization (p. 377).

Amazon ECS Service Auto Scaling supports the following types of scaling policies:

- **Target Tracking Scaling Policies (p. 321)**—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.

- **Step Scaling Policies (p. 326)**—Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, which vary based on the size of the alarm breach.

Target Tracking Scaling Policies

With target tracking scaling policies, you select a metric and set a target value. Amazon ECS creates and manages the CloudWatch alarms that trigger 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.

You can create multiple target tracking scaling policies for an Amazon ECS service, provided that each of them uses a different metric. The service scales based on the policy that provides the largest task capacity. This enables you to cover multiple scenarios and ensure that there is always enough capacity to process your application workloads.

To ensure application availability, the service scales out proportionally to the metric as fast as it can, but scales in more gradually.

Do not edit or delete the CloudWatch alarms that Amazon ECS manages for a target tracking scaling policy. Amazon ECS deletes the alarms automatically when you delete the target tracking scaling policy.
Considerations

Keep the following considerations in mind when creating a target tracking scaling policy:

- 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 Application 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. However, for a scalable target with small capacity, the actual metric data points might seem far from the target value. For a scalable target with larger capacity, adding or removing capacity causes less of a gap between the target value and the actual metric data points.
- We recommend that you scale based on metrics with a 1-minute frequency because that ensures a faster response to utilization changes. Scaling on metrics with a 5-minute frequency can result in slower response time and scaling on stale metric data.
- To ensure application availability, Application Auto Scaling scales out proportionally to the metric as fast as it can, but scales in more gradually.
- Do not edit or delete the CloudWatch alarms that Application Auto Scaling manages for a target tracking scaling policy. Application Auto Scaling deletes the alarms automatically when you delete the scaling policy.

Tutorial: Service Auto Scaling with Target Tracking

The following procedures help you to create an Amazon ECS cluster and a service that uses Application Auto Scaling to scale out (and in) using target tracking.

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. You can use these service utilization metrics to scale your service up to deal with high demand at peak times, and to scale your service down to reduce costs during periods of low utilization. For more information, see Service Utilization (p. 377).

In this tutorial, you create a cluster and a service (that runs behind an Elastic Load Balancing load balancer) using the Amazon ECS first run wizard. Then you configure Service Auto Scaling on the service with CloudWatch alarms that use the ECSServiceAverageCPUUtilization metric to scale your service up or down, depending on the current application load.

When the CPU utilization of your service rises above 75% (meaning that more than 75% of the CPU that is reserved for the service is being used), the scale-out alarm triggers Service Auto Scaling to add another task to your service to help out with the increased load. Conversely, when the CPU utilization of your service drops below 75%, the scale in alarm triggers a decrease in the service's desired count to free up those cluster resources for other tasks and services.

Prerequisites

This tutorial assumes that you have an AWS account, an IAM administrator with permissions to perform all of the actions described within, and an Amazon EC2 key pair in the current region. If you do not have these resources, or your are not sure, you can create them by following the steps in Setting Up with Amazon ECS (p. 7).

If you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, then you might need to add this permission. For information about checking your Amazon
ECS container instance role and attaching the managed IAM policy for container instances, see To check for the ecsInstanceRole in the IAM console (p. 417).

Step 1: Create a Cluster and a Service

After you have enabled CloudWatch metrics for your clusters and services, you can create a cluster and service using the Amazon ECS first-run wizard. The first-run wizard takes care of creating the necessary IAM roles and policies for this tutorial, an Auto Scaling group for your container instances, and a service that runs behind a load balancer. The wizard also makes the clean-up process much easier, because you can delete the entire AWS CloudFormation stack in one step.

For this tutorial, you create a cluster called service-autoscaling and a service called sample-webapp.

To create your cluster and service

1. Open the Amazon ECS console first run wizard at https://console.aws.amazon.com/ecs/home#/firstRun.
2. From the navigation bar, choose the US East (N. Virginia) region.
3. On Step 1: Container and Task, for Container definition, select sample-app.
4. For Task definition, leave all of the default options and choose Next.
5. On Step 2: Service, for Load balancer type, choose Application Load Balancer, Next.
   Important
   Application Load Balancers do incur costs while they exist in your AWS resources. For more information, see Elastic Load Balancing Pricing.
6. On Step 3: Cluster, for Cluster name, enter service-autoscaling and choose Next.
7. Review your choices and then choose Create.

You are directed to a Launch Status page that shows the status of your launch and describes each step of the process (this can take a few minutes to complete while your cluster resources are created and populated).
8. When your cluster and service are created, choose View service.

Step 2: Configure Service Auto Scaling

Now that you have launched a cluster and created a service in that cluster that is running behind a load balancer, you can configure Service Auto Scaling by creating scaling policies to scale your service out and in response to CloudWatch alarms.

To configure basic Service Auto Scaling parameters

1. On the Service: sample-app-service page, your service configuration should look similar to the image below, although the task definition revision and load balancer name are likely to be different. Choose Update to update your new service.
2. On the Update service page, choose Next step until you get to Step 3: Set Auto Scaling (optional).

3. For Service Auto Scaling, choose Configure Service Auto Scaling to adjust your service's desired count.

4. For Minimum number of tasks, enter 1 for the lower limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted below this amount.

5. For Desired number of tasks, this field is pre-populated with the value that you entered earlier. This value must be between the minimum and maximum number of tasks specified on this page. Leave this value at 1.

6. For Maximum number of tasks, enter 2 for the upper limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted above this amount.

7. For IAM role for Service Auto Scaling, choose an IAM role to authorize the Application Auto Scaling service to adjust your service's desired count on your behalf. If you have not previously created such a role, choose Create new role and the role is created for you. For future reference, the role that is created for you is called ecsAutoscaleRole. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 431).
To configure scaling policies for your service

These steps help you create scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a scale-out alarm to increase the desired count of your service, and a scale-in alarm to decrease the desired count of your service.

1. Choose **Add scaling policy** to configure your scaling policy.
2. On the **Add policy** page, update the following fields:
   a. For **Scaling policy type**, choose **Target tracking**.
   b. For **Policy name**, enter **TargetTrackingPolicy**.
   c. For **ECS service metric**, choose **ECSServiceAverageCPUUtilization**.
   d. For **Target value**, enter 75.
   e. For **Scale-out cooldown period**, enter 60. This is the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
   f. For **Scale-in cooldown period**, enter 60. This is the amount of time, in seconds, after a scale in activity completes before another scale in activity can start. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
   g. Choose **Save**.
3. Choose **Next step**.
4. Review all of your choices and then choose **Update Service**.
5. When your service status is finished updating, choose **View Service**.

**Step 3: Trigger a Scaling Activity**

After your service is configured with Service Auto Scaling, you can trigger a scaling activity by pushing your service's CPU utilization into the **ALARM** state. Because the example in this tutorial is a web application that is running behind a load balancer, you can send thousands of HTTP requests to your service (using the ApacheBench utility) to spike the service CPU utilization above the threshold amount. This spike should trigger the alarm, which in turn triggers a scaling activity to add one task to your service.

After the ApacheBench utility finishes the requests, the service CPU utilization should drop below your 25% threshold, triggering a scale in activity that returns the service's desired count to 1.

**To trigger a scaling activity for your service**

1. From your service's main view page in the console, choose the load balancer name to view its details in the Amazon EC2 console. You need the load balancer's DNS name, which should look something like **EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-east-1.elb.amazonaws.com**.
2. Use the ApacheBench (**ab**) utility to make thousands of HTTP requests to your load balancer in a short period of time.

   **Note**
   
   This command is installed by default on macOS, and it is available for many Linux distributions, as well. For example, you can install **ab** on Amazon Linux with the following command:

   ```bash
   $ sudo yum install -y httpd24-tools
   ```

   Run the following command, substituting your load balancer's DNS name.
4. In the left navigation pane, choose Alarms.
5. Wait for your `ab` HTTP requests to trigger the scale-out alarm in the CloudWatch console. You should see your Amazon ECS service scale out and add one task to your service's desired count.
6. Shortly after your `ab` HTTP requests complete (between 1 and 2 minutes), your scale in alarm should trigger and the scale in policy reduces your service's desired count back to 1.

Step 4: Cleaning Up

When you have completed this tutorial, you may choose to keep your cluster, Auto Scaling group, load balancer, and CloudWatch alarms. However, if you are not actively using these resources, you should consider cleaning them up so that your account does not incur unnecessary charges.

To delete your cluster and CloudWatch alarms

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the left navigation pane, choose Clusters.
3. On the Clusters page, choose the service-autoscaling cluster.
4. Choose Delete Cluster, Delete. It may take a few minutes for the cluster AWS CloudFormation stack to finish cleaning up.
5. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
6. Choose Alarms and select the alarms that begin with TargetTracking-service.
7. Choose Delete, Yes, Delete.

Step Scaling Policies

Your Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. Service Auto Scaling is available in all regions that support Amazon ECS.

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. You can use these service utilization metrics to scale your service up to deal with high demand at peak times, and to scale your service down to reduce costs during periods of low utilization. For more information, see Service Utilization (p. 377).

If your service contains tasks that use the EC2 launch type, you can also use Service Auto Scaling with Auto Scaling for Amazon EC2 on your Amazon ECS cluster to scale your cluster, and your service, as a result to the demand. For more information, see Tutorial: Scaling Container Instances with CloudWatch Alarms (p. 379). You can also use CloudWatch metrics published by other services, or custom metrics that are specific to your application. For example, a web service could increase the number of tasks based on Elastic Load Balancing metrics such as SurgeQueueLength, and a batch job could increase the number of tasks based on Amazon SQS metrics such as ApproximateNumberOfMessagesVisible.

Service Auto Scaling Required IAM Permissions

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. IAM users must have the appropriate permissions for these services before they can use Service Auto Scaling in the AWS
Management Console or with the AWS CLI or SDKs. In addition to the standard IAM permissions for creating and updating services, Service Auto Scaling requires the following permissions:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:*",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The Create Services (p. 448) and Update Services (p. 448) IAM policy examples show the permissions that are required for IAM users to use Service Auto Scaling in the AWS Management Console.

The Application Auto Scaling service needs permission to describe your ECS services and CloudWatch alarms, as well as permissions to modify your service's desired count on your behalf. You must create an IAM role (ecsAutoscaleRole) for your ECS services to provide these permissions and then associate that role with your service before it can use Application Auto Scaling. If an IAM user has the required permissions to use Service Auto Scaling in the Amazon ECS console, create IAM roles, and attach IAM role policies to them, then that user can create this role automatically as part of the Amazon ECS console create service (p. ) or update service (p. 350) workflows, and then use the role for any other service later (in the console or with the CLI or SDKs). You can also create the role by following the procedures in Amazon ECS Service Auto Scaling IAM Role (p. 431).

**Service Auto Scaling Concepts**

- The ECS 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 triggers a scale-out activity, Application 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 triggers a scale-in activity, Application Auto Scaling scales the desired count down 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, which is what processing capacity is supposed to be. This prevents excessive (runaway) scaling that could not be satisfied, for example, if there are not 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.
The Amazon ECS console's service creation and service update workflows support Service Auto Scaling. The Amazon ECS console handles the `ecsAutoScaleRole` and policy creation, provided that the IAM user who is using the console has the permissions described in Service Auto Scaling Required IAM Permissions (p. 326), and that they can create IAM roles and attach policies to them.

When you configure a service to use Service Auto Scaling in the console, your service is automatically registered as a scalable target with Application Auto Scaling so that you can configure scaling policies that scale your service up and down. You can also create and update the scaling policies and CloudWatch alarms that trigger them in the Amazon ECS console.

To create a new ECS service that uses Service Auto Scaling, see Creating a Service (p. 340).

To update an existing service to use Service Auto Scaling, see Updating a Service (p. 350).

**AWS CLI and SDK Experience**

You can configure Service Auto Scaling by using the AWS CLI or the AWS SDKs, but you must observe the following considerations.

- 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. For more information about these specific API operations, see the Amazon Elastic Container Service API Reference, the Amazon CloudWatch API Reference, and the Application Auto Scaling API Reference. For more information about the AWS CLI commands for these services, see the `ecs`, `cloudwatch`, and `application-autoscaling` sections of the AWS CLI Command Reference.

- Before your service can use Service Auto Scaling, you must register it as a scalable target with the Application Auto Scaling `RegisterScalableTarget` API operation.

- After your ECS service is registered as a scalable target, you can create scaling policies with the Application Auto Scaling `PutScalingPolicy` API operation to specify what should happen when your CloudWatch alarms are triggered.

- After you create the scaling policies for your service, you can create the CloudWatch alarms that trigger the scaling events for your service with the CloudWatch `PutMetricAlarm` API operation.

**Service Discovery**

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.

**Topics**

- Service Discovery Concepts (p. 328)
- Service Discovery Considerations (p. 329)
- Amazon ECS Console Experience (p. 330)
- Service Discovery Pricing (p. 330)
- Tutorial: Creating a Service Using Service Discovery (p. 330)

**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`.

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 using the Fargate launch type if they are using platform version v1.1.0 or later. For more information, see AWS Fargate Platform Versions (p. 36).
- The Create Service workflow in the Amazon ECS console only supports registering services into private DNS namespaces. When a AWS Cloud Map private DNS namespace is created, a Route 53 private hosted zone will be created automatically.
- Amazon ECS does not support registering services into public DNS 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 task definition your service task specifies 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 task definition that your service task specifies uses the `bridge` or `host` network mode, an SRV record is the only supported DNS record type. Create an SRV record for each service task. The SRV record must specify a container name and container port combination from the task definition.
Amazon ECS Console Experience

The service create and service update workflows in the Amazon ECS console supports service discovery.

To create a new Amazon ECS service that uses service discovery, see Creating a Service (p. 340).

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 are charged for those health checks.

Tutorial: Creating a Service Using Service Discovery

Service discovery has been integrated into the Create Service wizard in the Amazon ECS console. For more information, see Creating a Service (p. 340).

The following tutorial shows how to create an ECS service containing a Fargate task that uses service discovery with the AWS CLI.

Note
Fargate tasks are only supported in the following Regions:
<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
</tr>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
</tr>
<tr>
<td>EU (Ireland)</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>EU (London)</td>
<td>eu-west-2</td>
</tr>
<tr>
<td>EU (Frankfurt)</td>
<td>eu-central-1</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
</tr>
</tbody>
</table>

**Topics**

- Prerequisites (p. 331)
- Step 1: Create the Service Discovery Resources (p. 331)
- Step 2: Create the Amazon ECS Resources (p. 333)
- Step 3: Verify Service Discovery (p. 335)
- Step 4: Clean Up (p. 338)

**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 the AWS Command Line Interface.
- The steps in Setting Up with Amazon ECS (p. 7) have been completed.
- Your AWS user has the required permissions specified in the Amazon ECS First Run Wizard (p. 440) IAM policy example.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

**Step 1: Create the Service Discovery Resources**

Use the following steps to create your service discovery namespace and service discovery service.

To create the Service Discovery resources

1. Create a private service discovery namespace named tutorial within one of your existing VPCs:
aws servicediscovery create-private-dns-namespace --name tutorial --vpc vpc-abcd1234 --region us-east-1

Output:

```
{
  "OperationId": "h2qe3s6dxftvt7riu6lly2f6c3jlfh4-je6chs2e"
}
```

2. Using the `OperationId` from the previous output, verify that the private namespace was created successfully. Copy the namespace ID as it is used in subsequent commands.

```
aws servicediscovery get-operation --operation-id h2qe3s6dxftvt7riu6lly2f6c3jlfh4-je6chs2e
```

Output:

```
{
  "Operation": {
    "Id": "h2qe3s6dxftvt7riu6lly2f6c3jlfh4-je6chs2e",
    "Type": "CREATE_NAMESPACE",
    "Status": "SUCCESS",
    "CreateDate": 1519777852.502,
    "UpdateDate": 1519777856.086,
    "Targets": {
      "NAMESPACE": "ns-uejictsjen2i4eeg"
    }
  }
}
```

3. Using the `NAMESPACE` ID from the previous output, create a service discovery service named `myapplication`. Copy the service discovery service ID as it is used 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 --region us-east-1
```

Output:

```
{
  "Service": {
    "Id": "srv-utcrh6wavdkgqtk",
    "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

Use the following steps to create your Amazon ECS cluster, task definition, and service.

To create Amazon ECS resources

1. Create an Amazon ECS cluster named tutorial to use:

```
aws ecs create-cluster --cluster-name tutorial --region us-east-1
```

Output:

```
{
   "cluster": {
      "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/tutorial",
      "clusterName": "tutorial",
      "status": "ACTIVE",
      "registeredContainerInstancesCount": 0,
      "runningTasksCount": 0,
      "pendingTasksCount": 0,
      "activeServicesCount": 0,
      "statistics": []
   }  
}
```

2. 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 contents of the following task definition:

```
{
   "family": "tutorial-task-def",
   "networkMode": "awsvpc",
   "containerDefinitions": [ 
      { 
         "name": "sample-app",
         "image": "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>
httpd-foreground""]
      } 
   ]
}
```

API Version 2014-11-13

333
Then, register the task definition using the `fargate-task.json` file that you created:

```bash
aws ecs register-task-definition --cli-input-json file://fargate-task.json --region us-east-1
```

3. Create a file named `ecs-service-discovery.json` with the contents of the ECS service that you are going to create. This example uses the task definition created in the previous step. An `awsvpcConfiguration` is required because the example task definition uses the `awsvpc` network mode.

```json
{
    "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
}
```

Create your ECS service, specifying the Fargate launch type and the `LATEST` platform version, which supports service discovery:

```bash
aws ecs create-service --cli-input-json file://ecs-service-discovery.json --region us-east-1
```

Output:

```json
{  
    "service": {
        "serviceName": "ecs-service-discovery",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/tutorial",
        "loadBalancers": [],
        "serviceRegistries": [
            {
                "registryArn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkggqtk"
            }
        ]
    }
}
```
Step 3: Verify Service Discovery

You can verify that everything has been created properly by querying your service discovery information. After service discovery is configured, you can query it using either the AWS Cloud Map API operations or by using `dig` from within your VPC, as described below.
To verify service discovery configuration

1. Using the service discovery service ID, list the service discovery instances:

   ```
   aws servicediscovery list-instances --service-id srv-utcrh6wavadkggtk --region us-east-1
   ```

   Output:

   ```
   {
     "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. Using the service discovery namespace and service, use additional parameters to query the details about the service discovery instances:

   ```
   aws servicediscovery discover-instances --namespace-name tutorial --service-name myapplication --query-parameters ECS_CLUSTER_NAME=tutorial --region us-east-1
   ```

   Output:

   ```
   {
     "Instances": [
       {
         "InstanceId": "16becc26-8558-4af1-9fbd-f81be062a266",
         "NamespaceName": "tutorial",
         "ServiceName": "ecs-service-discovery",
         "HealthStatus": "HEALTHY",
         "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"
         }
       }
     ]
   }
   ```

3. The DNS records created in the Route 53 hosted zone for the service discovery service can be queried with the following AWS CLI commands.

   Using the namespace ID, get information about the namespace, which includes the Route 53 hosted zone ID:
aws servicediscovery get-namespace --id ns-uejictsjen2i4eeg --region us-east-1

Output:

```
{
  "Namespace": {
    "Id": "ns-uejictsjen2i4eeg",
    "Name": "tutorial",
    "Type": "DNS_PRIVATE",
    "Properties": {
      "DnsProperties": {
        "HostedZoneId": "Z35JQ42FDRYPLV"
      }
    },
    "CreateDate": 1519777852.502,
    "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
  }
}
```

4. Using the Route 53 hosted zone ID, get the resource record set for the hosted zone:

aws route53 list-resource-record-sets --hosted-zone-id Z35JQ42FDRYPLV --region us-east-1

Output:

```
{
  "ResourceRecordSets": [  
    {  
      "Name": "tutorial.",  
      "Type": "NS",  
      "TTL": 172800,  
      "ResourceRecords": [     
        {  
          "Value": "ns-1536.awsdns-00.co.uk."  
        },  
        {  
          "Value": "ns-0.awsdns-00.com."  
        },  
        {  
          "Value": "ns-1024.awsdns-00.org."  
        },  
        {  
          "Value": "ns-512.awsdns-00.net."  
        }  
      ]
    },  
    {  
      "Name": "tutorial.",  
      "Type": "SOA",  
      "TTL": 900,  
      "ResourceRecords": [     
        {  
          "Value": "ns-1536.awsdns-00.co.uk. awsdns-hostmaster.amazon.com. 1 7200 900 1209600 86400"  
        }  
      ]
    },  
    {  
```
5. You can also query the DNS using `dig` from an instance within your VPC with the following command:

   ```
   dig +short myapplication.tutorial
   ```

   Output:

   ```
   172.31.87.2
   ```

**Step 4: 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 service discovery instances and Amazon ECS resources**

1. Deregister the service discovery service instances:

   ```
   aws servicediscovery deregister-instance --service-id srv-utcrh6wavdkggqt --instance-id 16becc26-8558-4af1-9fbd-f81be062a266 --region us-east-1
   ```

   Output:

   ```
   {
     "OperationId": "xhu73bsertlyffhm3faq17kumsmx274n-jh0zimzv"
   }
   ```

2. Using the OperationId from the previous output, verify that the service discovery service instances were deregistered successfully:

   ```
   aws servicediscovery get-operation --operation-id xhu73bsertlyffhm3faq17kumsmx274n-jh0zimzv --region us-east-1
   ```

   Output:

   ```
   {
     "Operation": {
       "Id": "xhu73bsertlyffhm3faq17kumsmx274n-jh0zimzv",
       "Type": "DEREGISTER_INSTANCE",
       "Status": "SUCCESS",
       "CreateDate": 1525984073.707,
       "UpdateDate": 1525984076.426,
       "Targets": {
   ```
3. Delete the service discovery service:

```bash
aws servicediscovery delete-service --id srv-utcrh6wavdkggqtk --region us-east-1
```

4. Delete the service discovery namespace:

```bash
aws servicediscovery delete-namespace --id ns-uejictsjen2i4eeg --region us-east-1
```

Output:

```
{
  "OperationId": "c3ncqg1fsw4ibgj5baz6ktaho6cg4t-juhnzysj"
}
```

5. Using the OperationId from the previous output, verify that the service discovery namespace was deleted successfully:

```bash
aws servicediscovery get-operation --operation-id c3ncqg1fsw4ibgj5baz6ktaho6cg4t-juhnzysj --region us-east-1
```

Output:

```
{
  "Operation": {
    "Id": "c3ncqg1fsw4ibgj5baz6ktaho6cg4t-juhnzysj",
    "Type": "DELETE_NAMESPACE",
    "Status": "SUCCESS",
    "CreateDate": 1525984602.211,
    "UpdateDate": 1525984602.558,
    "Targets": {
      "NAMESPACE": "ns-rymlehshst7hhukh",
      "ROUTE_53_CHANGE_ID": "CJP2A2M86XWJO"
    }
  }
}
```

6. Update the Amazon ECS service so that the desired count is 0, which allows you to delete it:

```bash
aws ecs update-service --cluster tutorial --service ecs-service-discovery --desired-count 0 --force-new-deployment --region us-east-1
```

7. Delete the Amazon ECS service:

```bash
aws ecs delete-service --cluster tutorial --service ecs-service-discovery --region us-east-1
```

8. Delete the Amazon ECS cluster:

```bash
aws ecs delete-cluster --cluster tutorial --region us-east-1
```
Creating a Service

When you create an Amazon ECS service, you specify the basic parameters that define what makes up your service and how it should behave. These parameters create a service definition.

You can optionally configure additional features, such as an Elastic Load Balancing load balancer to distribute traffic across the containers in your service. For more information, see Service Load Balancing (p. 306). You must verify that your container instances can receive traffic from your load balancers. You can allow traffic to all ports on your container instances from your load balancer's security group to ensure that traffic can reach any containers that use dynamically assigned ports.

The following documents take you through each step of the create service wizard in the AWS Management Console.

Topics
- Step 1: Configuring Basic Service Parameters (p. 340)
- Step 2: Configure a Network (p. 342)
- Step 3: Configuring Your Service to Use a Load Balancer (p. 343)
- Step 4: (Optional) Configuring Your Service to Use Service Discovery (p. 347)
- Step 5: (Optional) Configuring Your Service to Use Service Auto Scaling (p. 348)
- Step 6: Review and Create Your Service (p. 350)

Step 1: Configuring Basic Service Parameters

All services require some basic configuration parameters that define the service, such as the task definition to use, which cluster the service should run on, how many tasks should be placed for the service, and so on. This is called the service definition. For more information about the parameters defined in a service definition, see Service Definition Parameters (p. 292).

This procedure covers creating a service with the basic service definition parameters that are required. After you have configured these parameters, you can create your service or move on to the procedures for optional service definition configuration, such as configuring your service to use a load balancer.

To configure the basic service definition parameters

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the Region that your cluster is in.
3. In the navigation pane, choose Task Definitions and select the task definition from which to create your service.
4. On the Task Definition name page, select the revision of the task definition from which to create your service.
5. Review the task definition, and choose Actions, Create Service.
6. On the Configure service page, fill out the following parameters accordingly:
   - **Launch type**: Choose whether your service should run tasks on Fargate infrastructure, or Amazon EC2 container instances that you maintain. For more information, see Amazon ECS Launch Types (p. 227).
   - **Platform version**: If you chose the Fargate launch type, then select the platform version to use.
   - **Cluster**: Select the cluster in which to create your service.
   - **Service name**: Type a unique name for your service.
   - **Service type**: Select a scheduling strategy for your service. For more information, see Service Scheduler Concepts (p. 290).
• Number of tasks: If you chose the REPLICA service type, type the number of tasks to launch and maintain on your cluster.

   Note
   If your launch type is EC2, and your task definition uses static host port mappings on your container instances, then you need at least one container instance with the specified port available in your cluster for each task in your service. This restriction does not apply if your task definition uses dynamic host port mappings with the bridge network mode. For more information, see portMappings (p. 199).

• If you are using the Rolling update deployment type, fill out the following parameters:

   • Minimum healthy percent: Specify a lower limit on the number of your service's tasks that must remain in the RUNNING state during a deployment, as a percentage of the service's desired number of tasks (rounded up to the nearest integer). For example, if your service has a desired number of four tasks and a minimum healthy percent of 50%, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that do not use a load balancer are considered healthy if they are in the RUNNING state. Tasks for services that do use a load balancer are considered healthy if they are in the RUNNING state and when the container instance on which it is hosted is reported as healthy by the load balancer. The default value for the minimum healthy percent is 50% in the console, and 100% with the AWS CLI or SDKs.

   • Maximum percent: Specify an upper limit on the number of your service's tasks that are allowed in the RUNNING or PENDING state during a deployment, as a percentage of the service's desired number of tasks (rounded down to the nearest integer). For example, if your service has a desired number of four tasks and a maximum percent value of 200%, 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%.

7. On the Deployments page, fill out the following parameters accordingly:

   • For Deployment type, choose whether your service should use a rolling update deployment or a blue/green deployment using AWS CodeDeploy. For more information, see Amazon ECS Deployment Types (p. 299).

   • If you selected the blue/green deployment type, for Service role for CodeDeploy choose the IAM service role for AWS CodeDeploy. For more information, see Amazon ECS CodeDeploy IAM Role (p. 428).

8. (Optional) If you selected the EC2 launch type and the REPLICA service type, for Task Placement, you can specify how tasks are placed using task placement strategies and constraints. Choose from the following options:

   • 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. See Amazon ECS Task Placement (p. 274) for examples.

For more information, see Amazon ECS Task Placement (p. 274).

9. In the Task tagging configuration section, complete the following steps:

   a. Select Enable ECS managed tags if you want Amazon ECS to automatically tag the tasks in the service with the Amazon ECS managed tags. For more information, see Tagging Your Amazon ECS Resources.

   b. For Propagate tags from, select one of the following:
• **Do not propagate** – This option will not propagate any tags to the tasks in the service.
• **Service** – This option will propagate the tags specified on your service to each of the tasks in the service.
• **Task Definitions** – This option will propagate the tags specified in the task definition of a task to the tasks in the service.

**Note**
If you specify a tag with the same key in the Tags section, it will override the tag propagated from either the service or the task definition.

10. In the Tags section, specify the key and value for each tag to associate with the task. For more information, see Tagging Your Amazon ECS Resources.

11. Choose Next step and navigate to Step 2: Configure a Network (p. 342).

**Step 2: Configure a Network**

If your service's task definition uses the `awsvpc` network mode, you must configure a VPC, subnet, and security group settings for your service.

If your service's task definition does not use the `awsvpc` network mode, you can move on to the next step, Step 3: Configuring Your Service to Use a Load Balancer (p. 343).

**To configure VPC and security group settings for your service**

1. If you have not done so already, follow the basic service configuration procedures in Step 1: Configuring Basic Service Parameters (p. 340).

2. For **Cluster VPC**, if you selected the EC2 launch type, choose the VPC in which your container instances reside. If you selected the Fargate launch type, select the VPC that the Fargate tasks should use. Ensure that the VPC you choose is not configured to require dedicated hardware tenancy, as that is not supported by Fargate tasks.

3. For **Subnets**, choose the available subnets for your service task placement.

   **Important**
   Only private subnets are supported for the `awsvpc` network mode. Because tasks do not receive public IP addresses, a NAT gateway is required for outbound internet access. Inbound internet traffic should be routed through a load balancer.

4. For **Security groups**, a security group has been created for your service's tasks, which allows HTTP traffic from the internet (0.0.0.0/0). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.

5. For **Auto-assign Public IP**, choose whether to have your tasks receive a public IP address. If you are using Fargate tasks, a public IP address needs to be assigned to the task's elastic network interface,. The network interface must have a route to the internet or a NAT gateway that can route requests to the internet, for the task to pull container images.

6. If you are configuring your service to use a load balancer or if you are using the green/blue deployment type, continue to Step 3: Configuring Your Service to Use a Load Balancer (p. 343). If you are not configuring your service to use a load balancer, you can choose **None** as the load balancer type and move on to the next section, Step 5: (Optional) Configuring Your Service to Use Service Auto Scaling (p. 348).
Step 3: Configuring Your Service to Use a Load Balancer

Services can be configured to use a load balancer to distribute incoming traffic to the tasks in your service. If your service is using the rolling update deployment type, this is optional. If your service is using the blue/green deployment type, then it is required to use either an Application Load Balancer or Network Load Balancer.

If you are not configuring your service to use a load balancer, you can choose **None** as the load balancer type and move on to the next section, Step 4: (Optional) Configuring Your Service to Use Service Discovery (p. 347).

If you have an available Elastic Load Balancing load balancer configured, you can attach it to your service with the following procedures, or you can configure a new load balancer. For more information, see Creating a Load Balancer (p. 311).

**Important**
Before following these procedures, you must create your Elastic Load Balancing load balancer resources.

Topics
- Configuring a Load Balancer for the Rolling Update Deployment Type (p. 343)
- Configuring a Load Balancer for the Blue/Green Deployment Type (p. 345)

Configuring a Load Balancer for the Rolling Update Deployment Type

If your service's tasks take a while to start and respond to Elastic Load Balancing health checks, you can specify a health check grace period of up to 2,147,483,647 seconds. During that time, the 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. This is only valid if your service is configured to use a load balancer.

**To configure a health check grace period**

1. If you have not done so already, follow the basic service configuration procedures in Step 1: Configuring Basic Service Parameters (p. 340).
2. For **Health check grace period**: Enter the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started.

To configure your service to use a load balancer, you must choose the load balancer type to use with your service.

**To choose a load balancer type**

1. If you have not done so already, follow the basic service creation procedures in Step 1: Configuring Basic Service Parameters (p. 340).
2. For **Load balancer type**, choose the load balancer type to use with your service:

   **Application Load Balancer**

   Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.
Network Load Balancer

Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing.

Classic Load Balancer

Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of the advanced features available to them.

3. For **Select IAM role for service**, choose **Create new role** to create a new role for your service, or select an existing IAM role to use for your service (by default, this is `ecsServiceRole`).

   **Important**
   If you choose to use an existing `ecsServiceRole` IAM role, you must verify that the role has the proper permissions to use Application Load Balancers and Classic Load Balancers. For more information, see Amazon ECS Service Scheduler IAM Role (p. 426).

4. For **ELB Name**, choose the name of the load balancer to use with your service. Only load balancers that correspond to the load balancer type you selected earlier are visible here.

5. The next step depends on the load balancer type for your service. If you've chosen an Application Load Balancer, follow the steps in To configure an Application Load Balancer (p. 344). If you've chosen a Network Load Balancer, follow the steps in To configure a Network Load Balancer (p. 345). If you've chosen a Classic Load Balancer, follow the steps in To configure a Classic Load Balancer (p. 345).

### To configure an Application Load Balancer

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose **Add to load balancer**.

2. For **Listener port**, choose the listener port and protocol of the listener that you created in Creating an Application Load Balancer (p. 312) (if applicable), or choose **create new** to create a new listener and then enter a port number and choose a port protocol for **Listener protocol**.

3. For **Target group name**, choose the target group that you created in Creating an Application Load Balancer (p. 312) (if applicable), or choose **create new** to create a new target group.

   **Important**
   If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), your target group must use `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.

4. (Optional) If you chose to create a new target group, complete the following fields as follows:

   - For **Target group name**, a default name is provided for you.
   - For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.
   - For **Path pattern**, if your listener does not have any existing rules, the default path pattern (`/`) is used. If your listener already has a default rule, then you must enter a path pattern that matches traffic that you want to have sent to your service's target group. For example, if your service is a web application called `web-app`, and you want traffic that matches `http://my-elb-url/web-app` to route to your service, then you would enter `/web-app/*` as your path pattern. For more information, see ListenerRules in the User Guide for Application Load Balancers.
   - For **Health check path**, enter the path to which the load balancer should send health check pings.

5. When you are finished configuring your Application Load Balancer, choose **Next step**.
To configure a Network Load Balancer

1. For Container to load balance, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose Add to load balancer.
2. For Listener port, choose the listener port and protocol of the listener that you created in Creating an Application Load Balancer (p. 312) (if applicable), or choose create new to create a new listener and then enter a port number and choose a port protocol for Listener protocol.
3. For Target group name, choose the target group that you created in Creating an Application Load Balancer (p. 312) (if applicable), or choose create new to create a new target group.

Important
If your service's task definition uses the awsvpc network mode (which is required for the Fargate launch type), your target group must use 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.

4. (Optional) If you chose to create a new target group, complete the following fields as follows:
   a. For Target group name, a default name is provided for you.
   b. For Target group protocol, enter the protocol to use for routing traffic to your tasks.
   c. For Health check path, enter the path to which the load balancer should send health check pings.
5. When you are finished configuring your Network Load Balancer, choose Next step.

To configure a Classic Load Balancer

1. The Health check port, Health check protocol, and Health check path fields are all pre-populated with the values you configured in Creating a Classic Load Balancer (p. 317) (if applicable). You can update these settings in the Amazon EC2 console.
2. For Container for ELB health check, choose the container to send health checks.
3. When you are finished configuring your Classic Load Balancer, choose Next step.

Configuring a Load Balancer for the Blue/Green Deployment Type

To configure your service that uses the blue/green deployment type to use a load balancer, you must use either an Application Load Balancer or a Network Load Balancer.

To choose a load balancer type

1. If you have not done so already, follow the basic service creation procedures in Step 1: Configuring Basic Service Parameters (p. 340).
2. For Load balancer type, choose the load balancer type to use with your service:

   Application Load Balancer
   Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.

   Network Load Balancer
   Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing.
We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of the advanced features available to them.

3. For **Load balancer name**, choose the name of the load balancer to use with your service. Only load balancers that correspond to the load balancer type you selected earlier are visible here.

4. The next step depends on the load balancer type for your service. If you've chosen an Application Load Balancer, follow the steps in *To configure an Application Load Balancer* (p. 344). If you've chosen a Network Load Balancer, follow the steps in *To configure a Network Load Balancer* (p. 345).

**To configure an Application Load Balancer for the blue/green deployment type**

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose *Add to load balancer*.

2. For **Production listener port**, choose the listener port and protocol of the listener that you created in *Creating an Application Load Balancer* (p. 312) (if applicable), or choose *create new* to create a new listener and then enter a port number and choose a port protocol for **Production listener protocol**.

3. (Optional) Select **Test listener** if you want to configure a listener port and protocol on your load balancer to test updates to your service before routing traffic to your new taskset. Complete the following step:

   • For **Test listener port**, choose the listener port and protocol of the listener that you want to test traffic over, or choose *create new* to create a new test listener and then enter a port number and choose a port protocol in **Test listener protocol**.

4. For blue/green deployments, two target groups are required. Each target group binds to a separate taskset in the deployment. Complete the following steps:

   a. For **Target group 1 name**, choose the target group that you created in *Creating an Application Load Balancer* (p. 312) (if applicable), or choose *create new* to create a new target group.

      **Important**
      If your service's task definition uses the *awsvpc* network mode (which is required for the Fargate launch type), your target group must use *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.

   b. (Optional) If you chose to create a new target group, complete the following fields as follows:

      • For **Target group name**, enter a name for your target group.

      • For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.

      • For **Path pattern**, if your listener does not have any existing rules, the default path pattern (/) is used. If your listener already has a default rule, then you must enter a path pattern that matches traffic that you want to have sent to your service's target group. For example, if your service is a web application called *web-app*, and you want traffic that matches http://my-elb-url/web-app to route to your service, then you would enter /web-app* as your path pattern. For more information, see **ListenerRules** in the *User Guide for Application Load Balancers*.

      • For **Health check path**, enter the path to which the load balancer should send health check pings.

   c. Repeat the steps for target group 2.

   d. When you are finished configuring your Application Load Balancer, choose **Next step**. Navigate to *Step 4: (Optional) Configuring Your Service to Use Service Discovery* (p. 347).
To configure a Network Load Balancer for the blue/green deployment type

1. For Container to load balance, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose Add to load balancer.

2. For Listener port, choose the listener port and protocol of the listener that you created in Creating an Application Load Balancer (p. 312) (if applicable), or choose create new to create a new listener and then enter a port number and choose a port protocol for Listener protocol.

3. For Target group name, choose the target group that you created in Creating an Application Load Balancer (p. 312) (if applicable), or choose create new to create a new target group.

   Important
   If your service's task definition uses the awsvpc network mode (which is required for the Fargate launch type), your target group must use 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.

4. (Optional) If you chose to create a new target group, complete the following fields as follows:
   • For Target group name, enter a name for your target group.
   • For Target group protocol, enter the protocol to use for routing traffic to your tasks.
   • For Health check path, enter the path to which the load balancer should send health check pings.

5. When you are finished configuring your Network Load Balancer, choose Next Step. Navigate to Step 4: (Optional) Configuring Your Service to Use Service Discovery (p. 347).

Step 4: (Optional) Configuring Your Service to Use Service Discovery

Your Amazon ECS service can optionally enable service discovery integration, which allows your service to be discoverable via DNS. For more information, see Service Discovery (p. 328).

If you are not configuring your service to use a service discovery, you can move on to the next section, Step 5: (Optional) Configuring Your Service to Use Service Auto Scaling (p. 348).

To configure service discovery

1. If you have not done so already, follow the basic service configuration procedures in Step 1: Configuring Basic Service Parameters (p. 340).

2. On the Configure network page, select Enable service discovery integration.

3. For Namespace, select an existing Amazon Route 53 namespace, if you have one, otherwise select create new private namespace.

4. If creating a new namespace, for Namespace name enter a descriptive name for your namespace. This is the name used for the Amazon Route 53 hosted zone.

5. For Configure service discovery service, select to either create a new service discovery service or select an existing one.

6. If creating a new service discovery service, for Service discovery name enter a descriptive name for your service discovery service. This is used as the prefix for the DNS records to be created.

7. Select Enable ECS task health propagation if you want health checks enabled for your service discovery service.

8. 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 DnsRecord.
• 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 the type A DNS record is selected, skip to the next step. If the type SRV is selected, specify either the port that the service can be found on or a container name and port combination to associate with the record.

9. For TTL, enter the resource record cache time to live (TTL), in seconds. This value determines how long a record set is cached by DNS resolvers and by web browsers.

10. Choose Next step to proceed and navigate to Step 5: (Optional) Configuring Your Service to Use Service Auto Scaling (p. 348).

Step 5: (Optional) Configuring Your Service to Use Service Auto Scaling

Your Amazon ECS service can optionally be configured to use Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms.

Amazon ECS Service Auto Scaling supports the following types of scaling policies:

• Target Tracking Scaling Policies (p. 321)—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.

• Step Scaling Policies (p. 326)—Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, which vary based on the size of the alarm breach.

For more information, see Service Auto Scaling (p. 321).

To configure basic Service Auto Scaling parameters

1. If you have not done so already, follow the basic service configuration procedures in Step 1: Configuring Basic Service Parameters (p. 340).

2. On the Set Auto Scaling page, select Configure Service Auto Scaling to adjust your service's desired count.

3. For Minimum number of tasks, enter the lower limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted below this amount.

4. For Desired number of tasks, this field is pre-populated with the value that you entered earlier. You can change your service's desired count at this time, but this value must be between the minimum and maximum number of tasks specified on this page.

5. For Maximum number of tasks, enter the upper limit of the number of tasks for Service Auto Scaling to use. Your service's desired count is not automatically adjusted above this amount.

6. For IAM role for Service Auto Scaling, choose an IAM role to authorize the Application Auto Scaling service to adjust your service's desired count on your behalf. If you have not previously created such a role, choose Create new role and the role is created for you. For future reference, the role that is created for you is called ecsAutoscaleRole. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 431).

7. The following procedures provide steps for creating either target tracking or step scaling policies for your service. Choose your desired scaling policy type.
These steps help you create target tracking scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a scale-out alarm to increase the desired count of your service, and a scale-in alarm to decrease the desired count of your service.

**To configure target tracking scaling policies for your service**

1. For **Scaling policy type**, choose **Target tracking**.
2. For **Policy name**, enter a descriptive name for your policy.
3. For **ECS service metric**, choose the metric to track.
4. For **Target value**, enter the metric value that the policy should maintain.
5. For **Scale-out cooldown period**, enter the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
6. 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. During this time, resources that have been launched do not contribute to the Auto Scaling group metrics.
7. (Optional) To disable the scale-in actions for this policy, choose **Disable scale-in**. This allows you to create a separate scaling policy for scale-in later.
8. Choose **Next step**.

These steps help you create step scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a **Scale out** alarm to increase the desired count of your service, and a **Scale in** alarm to decrease the desired count of your service.

**To configure step scaling policies for your service**

1. For **Scaling policy type**, choose **Step scaling**.
2. For **Policy name**, enter a descriptive name for your policy.
3. For **Execute policy when**, select the CloudWatch alarm to use to scale your service up or down.
   
   You can use an existing CloudWatch alarm that you have previously created, or you can choose to create a new alarm. The **Create new alarm** workflow allows you to create CloudWatch alarms that are based on the CPUUtilization and MemoryUtilization of the service that you are creating. To use other metrics, you can create your alarm in the CloudWatch console and then return to this wizard to choose that alarm.
4. (Optional) If you've chosen to create a new alarm, complete the following steps.
   
   a. For **Alarm name**, enter a descriptive name for your alarm. For example, if your alarm should trigger when your service CPU utilization exceeds 75%, you could call the alarm `service_name-cpu-gt-75`.
   
   b. For **ECS service metric**, choose the service metric to use for your alarm. For more information, see **Service Utilization (p. 377)**.
   
   c. For **Alarm threshold**, enter the following information to configure your alarm:
      
      • Choose the CloudWatch statistic for your alarm (the default value of **Average** works in many cases). For more information, see **Statistics** in the **Amazon CloudWatch User Guide**.
      • Choose the comparison operator for your alarm and enter the value that the comparison operator checks against (for example, > and 75).
      • Enter the number of consecutive periods before the alarm is triggered and the period length. For example, two consecutive periods of 5 minutes would take 10 minutes before the alarm triggered. Because your Amazon ECS tasks can scale up and down quickly, consider using a low number of consecutive periods and a short period duration to react to alarms as soon as possible.
d. Choose **Save**.

5. For **Scaling action**, enter the following information to configure how your service responds to the alarm:

   - Choose whether to add to, subtract from, or set a specific desired count for your service.
   - If you chose to add or subtract tasks, enter the number of tasks (or percent of existing tasks) to add or subtract when the scaling action is triggered. If you chose to set the desired count, enter the desired count that your service should be set to when the scaling action is triggered.
   - (Optional) If you chose to add or subtract tasks, choose whether the previous value is used as an integer or a percent value of the existing desired count.
   - Enter the lower boundary of your step scaling adjustment. By default, for your first scaling action, this value is the metric amount where your alarm is triggered. For example, the following scaling action adds 100% of the existing desired count when the CPU utilization is greater than 75%.

6. (Optional) You can repeat Step 5 (p. 350) to configure multiple scaling actions for a single alarm (for example, to add one task if CPU utilization is between 75-85%, and to add two tasks if CPU utilization is greater than 85%).

7. (Optional) If you chose to add or subtract a percentage of the existing desired count, enter a minimum increment value for **Add tasks in increments of N task(s)**.

8. For **Cooldown period**, enter the number of seconds between scaling actions.

9. Repeat Step 1 (p. 349) through Step 8 (p. 350) for the **Scale in** policy and choose **Save**.

10. Choose **Next step** to proceed and navigate to **Step 6: Review and Create Your Service** (p. 350).

### Step 6: Review and Create Your Service

After you have configured your basic service definition parameters and optionally configured your service's networking, load balancer, service discovery, and automatic scaling, you can review your configuration. Then, choose **Create Service** to finish creating your service.

**Note**

After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

### Updating a Service

You can update a running service to change the number of tasks that are maintained by a service, which task definition is used by the tasks, or if your tasks are using the Fargate launch type, you can change the platform version your service uses. If you have an application that needs more capacity, you can scale up your service. If you have unused capacity to scale down, you can reduce the number of desired tasks in your service and free up resources.

If you have updated the Docker image of your application, you can create a new task definition with that image and deploy it to your service.
Note
If your updated Docker image uses the same tag as what is in the existing task definition for your service (for example, my_image:latest), you do not need to create a new revision of your task definition. You can update the service using the procedure below, keep the current settings for your service, and select force new deployment. The new tasks launched by the deployment pull the current image/tag combination from your repository when they start. The force new deployment option is also used when updating a Fargate task to use a more current platform version when you specify LATEST. For example, if you specified LATEST and your running tasks are using the 1.0.0 platform version and you want them to relaunch using a newer platform version.

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 is using 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. This parameter enables you 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%, the scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks. Tasks for services that do not use a load balancer are considered healthy if they are in the RUNNING state. Tasks for services that do use a load balancer are considered healthy 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%.

If a service is using 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 RUNNING or PENDING 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. This parameter enables you 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%, the scheduler may start four new tasks before stopping the four older tasks. That's provided that the cluster resources required to do this are available. The default value for the maximum percent is 200%.

If a service is using the blue/green (CODE_DEPLOY) deployment type and tasks that use the EC2 launch type, the minimum healthy percent and maximum percent values are set to the default values. They are only used to define the lower and 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. If the tasks in the service use the Fargate launch type, the minimum healthy percent and maximum percent values are not used. They are currently visible when describing your service.

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 SIGHUP 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 SIGHUP signal is sent. The service scheduler starts and stops tasks as defined by your minimum healthy percent and maximum percent settings.

Important
If you are changing the ports used by containers in a task definition, you may need to update your container instance security groups to work with the updated ports.

If your service uses a load balancer, the load balancer configuration defined for your service when it was created cannot be changed. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.
To change the load balancer name, the container name, or the container port associated with a service load balancer configuration, you must create a new service. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

**To update a running service**

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the Region that your cluster is in.
3. In the navigation pane, choose Clusters.
4. On the Clusters page, select the name of the cluster in which your service resides.
6. Check the box to the left of the service to update and choose Update.
7. On the Configure service page, your service information is pre-populated. Change the task definition, platform version, deployment configuration, or number of desired tasks (or any combination of these) and choose Next step.

   **Note**

   To have your service use a newly updated Docker image with the same tag as in the existing task definition (for example, my_image:latest) or keep the current settings for your service, select Force new deployment. The new tasks launched by the deployment pull the current image/tag combination from your repository when they start. The Force new deployment option is also used when updating a Fargate task to use a more current platform version when you specify LATEST. For example, if you specified LATEST and your running tasks are using the 1.0.0 platform version and you want them to relaunch using a newer platform version.

8. On the Configure deployments page, if your service is using the blue/green deployment type, the components of your service deployment is pre-populated. Confirm the following settings.

   a. For Application name, choose the CodeDeploy application of which your service is a part.
   b. For Deployment group name, choose the CodeDeploy deployment group of which your service is a part.
   c. Select the deployment lifecycle event hooks and the associated Lambda functions to execute as part of the new revision of the service deployment. The available lifecycle hooks are:

      - **BeforeInstall** – Use this deployment lifecycle event hook to invoke a Lambda function before the replacement task set is created. The result of the Lambda function at this lifecycle event does not trigger a rollback.
      - **AfterInstall** – Use this deployment lifecycle event hook to invoke a Lambda function after the replacement task set is created. The result of the Lambda function at this lifecycle event can trigger a rollback.
      - **BeforeAllowTraffic** – Use this deployment lifecycle event hook to invoke a Lambda function before the production traffic has been rerouted to the replacement task set. The result of the Lambda function at this lifecycle event can trigger a rollback.
      - **AfterAllowTraffic** – Use this deployment lifecycle event hook to invoke a Lambda function after the production traffic has been rerouted to the replacement task set. The result of the Lambda function at this lifecycle event can trigger a rollback.

   For more information about lifecycle hooks, see AppSpec 'hooks' Section in the AWS CodeDeploy User Guide.

10. On the Configure network page, your network information is pre-populated. In the Load balancing section, if your service is using the blue/green deployment type, select the listeners to associate with the target groups. Change the health check grace period (if desired) and choose Next step.
Deleting a Service

You can delete an Amazon ECS service using the console. Before deletion, the service is automatically scaled down to zero. If you have a load balancer or service discovery resources associated with the service, they 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. To delete your service discovery resources, follow the procedure below.

To delete an Amazon ECS service

Use the following procedure to delete an Amazon ECS service.

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar, select the Region that your cluster is in.
3. In the navigation pane, choose Clusters and select the name of the cluster in which your service resides.
5. Check the box to the left of the service to update and choose Delete.
6. Confirm the service deletion by entering the text phrase and choose Delete.

To delete the service discovery resources (AWS CLI)

To delete the remaining service discovery resources, you can use the AWS CLI to delete the service discovery service and service discovery namespace.

1. Ensure that 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.
2. Retrieve the ID of the service discovery service to delete.

   ```sh
   aws servicediscovery list-services --region <region_name>
   ```

   **Note**
   
   If no service discovery service is returned, continue to step 4.
3. Using the service discovery service ID from the previous output, delete the service.

   ```sh
   aws servicediscovery delete-service --id <service_discovery_service_id> --region <region_name>
   ```
4. Retrieve the ID of the service discovery namespace to delete.

   ```sh
   aws servicediscovery list-namespaces --region <region_name>
   ```
5. Using the service discovery namespace ID from the previous output, delete the namespace.
Service Throttle Logic

The Amazon ECS service scheduler includes logic that throttles how often service tasks are launched if they repeatedly fail to start.

If tasks for an ECS service repeatedly fail to enter the **RUNNING** state (progressing directly from **PENDING** to **STOPPED**), then the time between subsequent restart attempts is incrementally increased up to 15 minutes. This maximum period is subject to change in the future and should not be considered permanent. This behavior reduces the effect that unstartable tasks have on your Amazon ECS cluster resources or Fargate infrastructure costs. If your service triggers the throttle logic, you receive the following service event message (p. 594):

```
(service service-name) is unable to consistently start tasks successfully.
```

Amazon ECS does not ever stop a failing service from retrying, nor does it attempt to modify it in any way other than increasing the time between restarts. The service throttle logic does not provide any user-tunable parameters.

If you update your service to use a new task definition, your service returns to a normal, non-throttled state immediately. For more information, see Updating a Service (p. 350).

The following are some common causes that trigger this logic:

- A lack of resources with which to host your task, such as ports, memory, or CPU units in your cluster. In this case, you also see the insufficient resource service event message (p. 592).
- The Amazon ECS container agent is unable to pull your task Docker image. This could be due to a bad container image name, image, or tag, or a lack of private registry authentication or permissions. In this case, you also see `CannotPullContainerError` in your stopped task errors (p. 590).
- Insufficient disk space on your container instance to create the container. In this case, you also see `CannotCreateContainerError` in your stopped task errors (p. 590). For more information, see `CannotCreateContainerError: API error (500): devmapper` (p. 596).

**Important**

Tasks that are stopped after they reach the **RUNNING** state do not trigger the throttle logic or the associated service event message. For example, if failed Elastic Load Balancing health checks for a service cause a task to be flagged as unhealthy, and Amazon ECS deregisters it and kills the task, this does not trigger the throttle. Even if a task's container command immediately exits with a non-zero exit code, the task has already moved to the **RUNNING** state. Tasks that fail immediately due to command errors do not trigger the throttle or the service event message.
Resources and Tags

Amazon ECS resources, including task definitions, clusters, tasks, services, and container instances, are assigned an Amazon Resource Name (ARN) and a unique resource identifier (ID). These resources can be tagged with values that you define, to help you organize and identify them.

The following topics describe resources and tags, and how you can work with them.

Contents
- Amazon Resource Names (ARNs) and IDs (p. 355)
- Tagging Your Amazon ECS Resources (p. 359)
- Amazon ECS Usage Reports (p. 365)

Amazon Resource Names (ARNs) and IDs

When Amazon ECS resources are created, we assign each resource a unique Amazon Resource Name (ARN) and resource identifier (ID). If you are using 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 are using the `stop-task` AWS CLI command to stop a task, you must specify the task ARN or ID in the command.

We're gradually introducing a new ARN and resource ID format for Amazon ECS tasks, services and container instances. The following sections describe how the formats are changing. For more information on the transition to the new formats, see Amazon Elastic Container Service FAQ.

Amazon Resource Name (ARN) Format

Some resources have a friendly name (for example, a service named `production`). However, sometimes you are required to specify a resource using the Amazon Resource Name (ARN) format. We're gradually introducing a new ARN format for Amazon ECS tasks, services and container instances which includes the cluster name. For details on how to opt in to the new ARN format, see Working with Account Settings (p. 356).

**Note**

The new ARN format is not available in the GovCloud (US-East) region.

The following table shows both the current (old) format and the new format for each resource type:

<table>
<thead>
<tr>
<th>Resource type</th>
<th>ARN</th>
</tr>
</thead>
</table>
| Amazon ECS service | Old: arn:aws:ecs:region:aws_account_id:service/service-name  
| Amazon ECS task | Old: 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. We're gradually introducing shorter length IDs for Amazon ECS tasks and container instances. The length of the ID was in a 36-character format; the new IDs are in a 32-character format that will not include any hyphens. For details on how to opt in to the new resource ID format, see Working with Account Settings (p. 356).

Note
The new resource ID format is not available in the GovCloud (US-East) region.

Timeline

The new formats have an opt-in period, during which you can choose to accept the new formats. The following lists the important dates related to this change:

- Now until March 31, 2019 - The option to opt-in to the new Amazon Resource Name (ARN) and resource ID formats begins. The ability to opt-in and opt-out is provided on a per-region basis. Any new accounts created will be opted-out by default.
- April 1, 2019 - December 31, 2019 - All new accounts will be opted-in by default. The ability to opt-in and opt-out will continue to be available on a per-region basis.
- January 1, 2020 - All accounts will be opted-in by default. All new resources created will receive the new format.

You can opt-in or opt-out of the new Amazon Resource Name (ARN) and resource ID format at any time during the opt-in period. After you have opted in, any new resources that you create are created with the new format.

Note
A resource ID does not change after it's created. Therefore, opting in or out of the new format during the opt-in period does not affect your existing resource IDs.

Working with Account Settings

For each Region, you can opt in or opt out of the new ARN and resource ID format at the account level or for a specific IAM user or role. The following are supported scenarios:

- An IAM user or role can opt in or opt out for their invididual user account.
- An IAM user or role can set the default opt in or opt out setting for all users on the account.
- The root user has the ability to opt in or opt out any specific IAM role or user on the account. If the account setting for the root user is changed, it sets the default setting for all the IAM users and roles for which no individual account setting has been set.

The opt in and opt out account setting must be set for each Amazon ECS resource separately. The ARN and resource ID format of a resource is defined by the opt-in status of the IAM user or role that created the resource.

Only resources launched after opting in will receive the new ARN and resource ID format. All existing resources keep their ARN and resource ID and aren't affected. For Amazon ECS services and tasks to transition to the new format, the service or task must be recreated. To transition a container instance to the new format, the container instance must be drained and a new container instance 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 IAM user who created the service has opted in for the new format for tasks.

Topics
• Viewing Account Settings (p. 357)
• Modifying Account Settings (p. 357)

Viewing Account Settings

You can use the AWS Management Console and AWS CLI tools to view the resource types that support the new ARN and ID formats.

To view your account settings using the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation bar at the top of the screen, select the Region for which to view your account settings.
3. From the dashboard, under Clusters, choose Configure ECS ARN setting.
4. On the Amazon ECS ARN and resource ID settings section, you can view your account settings for each resource type for the authenticated IAM user and role.

To view your account settings using the command line

Use one of the following commands to view your account settings.

• list-account-settings (AWS CLI)

```bash
aws ecs list-account-settings --effective-settings --region us-east-1
```

• Get-ECSAccountSetting (AWS Tools for Windows PowerShell)

```powershell
Get-ECSAccountSetting -EffectiveSetting true -Region us-east-1
```

To view the account settings for a specific IAM user or IAM role using the command line

Use one of the following commands and specify the ARN of an IAM user, IAM role, or root account user in the request to view their account settings.

• list-account-settings (AWS CLI)

```bash
aws ecs list-account-settings --principal-arn arn:aws:iam::aws_account_id:user/principalName --effective-settings --region us-east-1
```

• Get-ECSAccountSetting (AWS Tools for Windows PowerShell)

```powershell
Get-ECSAccountSetting -PrincipalArn arn:aws:iam::aws_account_id:user/principalName -EffectiveSetting true -Region us-east-1
```

Modifying Account Settings

You can use the AWS Management Console and AWS CLI tools to modify account settings for resource types that are still within their opt-in period.

To modify account settings using the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation bar at the top of the screen, select the region for which to modify the longer ID settings.
3. From the dashboard, choose **Configure ECS ARN setting**.
4. To enable the new ARN and ID format for each supported resource type, select the option in the **My IAM user/role opt-in setting** column. This will change the opt in setting for the authenticated IAM user.
5. On the confirmation screen, choose **Confirm** to save the selection.

**To modify the default account settings for all IAM users or roles on your account using the command line**

Use one of the following commands to modify the default account setting for all IAM users or roles on your account. These changes apply to the entire AWS account unless an IAM user or role explicitly overrides these settings for themselves.

- **put-account-setting-default** (AWS CLI)
  ```bash
  aws ecs put-account-setting-default --name serviceLongArnFormat --value enabled --region us-east-2
  ```

  You can also use the command to modify the account settings for all tasks (taskLongArnFormat) and container instances (containerInstanceLongArnFormat). To do this, replace the name parameter with the corresponding resource type.

- **Write-ECSAccountSetting** (AWS Tools for Windows PowerShell)
  ```ps
  Write-ECSAccountSettingDefault -Name serviceLongArnFormat -Value enabled -Force
  ```

**To modify the account settings for your IAM user account using the command line**

Use one of the following commands to modify the account settings for your IAM user. If you're using these commands as the root user, changes apply to the entire AWS account unless an IAM user or role explicitly overrides these settings for themselves.

- **put-account-setting** (AWS CLI)
  ```bash
  aws ecs put-account-setting --name serviceLongArnFormat --value enabled --region us-east-1
  ```

  You can also use the command to modify the account settings for all tasks (taskLongArnFormat) and container instances (containerInstanceLongArnFormat). To do this, replace the name parameter with the corresponding resource type.

- **Write-ECSAccountSetting** (AWS Tools for Windows PowerShell)
  ```ps
  Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -Force
  ```

**To modify the account settings for a specific IAM user or IAM role using the command line**

Use one of the following commands and specify the ARN of an IAM user, IAM role, or root user in the request to modify the account settings for a specific IAM user or IAM role.

- **put-account-setting** (AWS CLI)
Tagging Your Amazon ECS Resources

To help you manage your Amazon ECS tasks, services, task definitions, clusters, and container instances, you can optionally assign your own metadata to each resource in the form of tags. This topic describes tags and shows you how to create them.

Important
To use this feature, it requires that you opt-in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see Amazon Resource Names (ARNs) and IDs (p. 355).

Contents
- Tag Basics (p. 359)
- Tagging Your Resources (p. 360)
- Tag Restrictions (p. 361)
- Tagging Your Resources for Billing (p. 361)
- Working with Tags Using the Console (p. 361)
- Working with Tags Using the CLI or API (p. 363)

Tag Basics

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value, both of which you define.

Tags enable you to categorize your AWS 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 you've assigned to it. For example, you could define a set of tags for your account's Amazon ECS container instances that helps you track each container instance's owner and stack level.

We recommend that you devise a set of tag keys that meets your needs for each resource type. Using a consistent set of tag keys makes it easier for you to manage 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. Also, tags are not automatically assigned to your resources. 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 old value. If you delete a resource, any tags for the resource are also deleted.

You can work with tags using the AWS Management Console, the AWS CLI, and the Amazon ECS API.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.

**Tagging Your Resources**

You can tag new or existing Amazon ECS tasks, services, task definitions, and clusters.

If you're using the Amazon ECS console, you can apply tags to new resources when they are created or existing resources by using the Tags tab on the relevant resource page at any time. The Propagate tags from option can be used when running a task to copy the tags from the task definition to the task or when creating a service to copy the tags from either the service or the task definition to the tasks in the service.

If you're using the Amazon ECS API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the tags parameter on the relevant API action or use the TagResource API action to apply tags to existing resources. For more information, see TagResource. The propagateTags parameter can be used when running a task to copy the tags from the task definition to the task or when creating a service to copy the tags from either the service or the task definition to the tasks in the service. For more information, see RunTask and CreateService.

Additionally, some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, we roll back the resource 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 at the time of creation, you can eliminate the need to run custom tagging scripts after resource creation.

The following table describes the Amazon ECS resources that can be tagged, and the resources that can be tagged on creation.

**Tagging Support for Amazon ECS Resources**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Supports tags</th>
<th>Supports tag propagation</th>
<th>Supports tagging on creation (Amazon ECS API, AWS CLI, AWS SDK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon ECS tasks</td>
<td>Yes</td>
<td>Yes, from the task definition.</td>
<td>Yes</td>
</tr>
<tr>
<td>Amazon ECS services</td>
<td>Yes</td>
<td>Yes, from either the task definition or the service to the tasks in the service.</td>
<td>Yes</td>
</tr>
<tr>
<td>Amazon ECS task definitions</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Amazon ECS clusters</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Amazon ECS container instances</td>
<td>Yes</td>
<td>Yes, from the Amazon EC2 instance. For more information, see Adding Tags to a Container Instance (p. 362).</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Tag Restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are: letters, numbers, and spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case-sensitive.
- Don't use the aws: prefix for either keys or values; it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags per resource limit.

Tagging Your Resources for Billing

When enabling Amazon ECS managed tags, Amazon ECS will automatically tag all newly launched tasks with the cluster name. For tasks that belong to a service, they will be tagged with the service name as well. These managed tags are helpful when reviewing cost allocation after enabling them in your Cost & Usage Report. For more information, see Amazon ECS Usage Reports (p. 365).

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 and Cost Management User Guide.

Important

To use this feature, it requires that you opt-in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see Amazon Resource Names (ARNs) and IDs (p. 355).

Note

If you've just enabled reporting, data for the current month is available for viewing after 24 hours.

Working with Tags Using the Console

Using the Amazon ECS console, you can manage the tags associated with new or existing tasks, services, task definitions, clusters, or container instances.

When you select a resource-specific page in the Amazon ECS console, it displays a list of those resources. For example, if you select Clusters from the navigation pane, the console displays a list of Amazon ECS clusters. When you select a resource from one of these lists (for example, a specific cluster), if the resource supports tags, you can view and manage its tags on the Tags tab.

Contents

- Adding Tags on an Individual Resource During Launch (p. 362)
- Adding and Deleting Tags on an Individual Resource (p. 362)
- Adding Tags to a Container Instance (p. 362)
Adding Tags on an Individual Resource During Launch

The following resources allow you to specify tags when you create the resource.

<table>
<thead>
<tr>
<th>Task</th>
<th>Console</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run one or more tasks.</td>
<td>Running Tasks (p. 269)</td>
</tr>
<tr>
<td>Create a service.</td>
<td>Creating a Service (p. 340)</td>
</tr>
<tr>
<td>Register a task definition.</td>
<td>Creating a Task Definition (p. 188)</td>
</tr>
<tr>
<td>Create a cluster.</td>
<td>Creating a Cluster (p. 55)</td>
</tr>
<tr>
<td>Run one or more container instances.</td>
<td>Launching an Amazon ECS Container Instance (p. 101)</td>
</tr>
</tbody>
</table>

Adding and Deleting Tags on an Individual Resource

Amazon ECS allows you to add or delete tags associated with your clusters, services, tasks, and task definitions directly from the resource's page. For information about tagging your container instances, see Adding Tags to a Container Instance (p. 362).

To add a tag to an individual resource

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the region to use.
3. In the navigation pane, select a resource type (for example, Clusters).
4. Select the resource from the resource list and choose Tags, Edit.
5. In the Edit Tags dialog box, specify the key and value for each tag, and then choose Save.

To delete a tag from an individual resource

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the region to use.
3. In the navigation pane, choose a resource type (for example, Clusters).
4. Select the resource from the resource list and choose Tags, Edit.
5. On the Edit Tags page, select the Delete icon for each tag you want to delete, and choose Save.

Adding Tags to a Container Instance

You can associate tags with your container instances using one of the following methods:

- **Method 1** – When creating your 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 Container Instances with Amazon EC2 User Data (p. 105).
  
  **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 would associate tags with your container instance:

```bash
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

• Method 2 – When creating your container instance using the Amazon EC2 API, CLI, or console, specify tags using the `TagSpecification.N` parameter and then pass user data to the instance using the container agent configuration parameter `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` which will 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 an Amazon EC2 instance, as well as register the instance with a cluster named `MyCluster`:

```bash
#!/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`

An example inline policy adding the permissions is shown below:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeTags"
      ],
      "Resource": "*"
    }
  ]
}
```

## Working with Tags Using the CLI or API

Use the following to add, update, list, and delete the tags for your resources. The corresponding documentation provides examples.

### Tagging Support for Amazon ECS Resources

<table>
<thead>
<tr>
<th>Task</th>
<th>AWS CLI</th>
<th>API Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add or overwrite one or more tags.</td>
<td><code>tag-resource</code></td>
<td><code>TagResource</code></td>
</tr>
</tbody>
</table>
The following examples show how to tag or untag resources using the AWS CLI.

**Example 1: Tag an existing cluster**

The following command tags an existing cluster.

```bash
aws ecs tag-resource --resource-arn resource_ARN --tags key=stack,value=dev
```

**Example 2: Untag an existing cluster**

The following command deletes a tag from an existing cluster.

```bash
aws ecs untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

**Example 3: List tags for a resource**

The following command lists the tags associated with an existing resource.

```bash
aws ecs list-tags-for-resource --resource-arn resource_ARN
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.

<table>
<thead>
<tr>
<th>Task</th>
<th>AWS CLI</th>
<th>AWS Tools for Windows PowerShell</th>
<th>API Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run one or more tasks.</td>
<td>run-task</td>
<td>Start-ECSTask</td>
<td>RunTask</td>
</tr>
<tr>
<td>Create a service.</td>
<td>create-service</td>
<td>New-ECSService</td>
<td>CreateService</td>
</tr>
<tr>
<td>Register a task definition.</td>
<td>register-task-definition</td>
<td>Register-ECSTaskDefinition</td>
<td>RegisterTaskDefinition</td>
</tr>
<tr>
<td>Create a cluster.</td>
<td>create-cluster</td>
<td>New-ECSCluster</td>
<td>CreateCluster</td>
</tr>
<tr>
<td>Run one or more container instances.</td>
<td>run-instances</td>
<td>New-EC2Instance</td>
<td>RunInstances</td>
</tr>
</tbody>
</table>

The following examples demonstrate how to apply tags when you create resources.

**Example 1: Create a cluster and apply a tag**

The following command creates a cluster named devcluster and adds a tag with key team and value devs.

```bash
aws ecs create-cluster --cluster-name devcluster --tags key=team,value=devs
```

**Example 2: Create a service and apply a tag**

The following command creates a service named application and adds a tag with key stack and value dev.
Example 3: Create a service with tags and propagate the tags to the tasks in the service

The `--propagateTags` parameter can be used to copy the tags from either a task definition or a service to the tasks in a service. The following command creates a service with tags and propagates them to the tasks in that service.

```
aws ecs create-service --service-name application --task-definition task-def-app --tags key=stack,value=dev --propagateTags Service
```

Amazon ECS Usage Reports

AWS provides a free reporting tool called Cost Explorer that enables you to analyze the cost and usage of your Amazon ECS resources.

Cost Explorer is a free tool that you can use to view charts of your usage and costs. You can view data from the last 13 months, and forecast how much you are 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, 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 & Usage Report shows usage across all of your Amazon ECS tasks. The metering data includes vCPU-Hours and memory GB-Hours for each task that was run. For tasks using the Fargate launch type, you will also see the cost associated with those tasks. For tasks using the EC2 launch type, the tasks will not have a cost associated with them, but you can use the the vCPU and/or memory usage to allocate the cost of your underlying cluster of Amazon EC2 instances. You can also use the Amazon ECS managed tags to identify the service or cluster that each task belongs to. For more information, see Tagging Your Resources for Billing (p. 361).

**Important**

The metering data is only viewable for tasks launched on or after November 16, 2018. Tasks launched prior to this date will not show metering data.

Here's an example of some of the fields you can sort cost allocation data by when using Cost Explorer:

- Cluster name
- Service name
- Resource tags
- Launch type
- Region
- Usage type

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 enables you to privately access Amazon ECS APIs by using private IP addresses. 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.

You're not required to configure PrivateLink, but we recommend it. For more information about PrivateLink and VPC endpoints, see Accessing Services Through AWS PrivateLink.

Topics
- Considerations for Amazon ECS VPC Endpoints (p. 366)
- Creating the VPC Endpoints for Amazon ECS (p. 366)

Considerations for Amazon ECS VPC Endpoints

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 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.
- 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.
- Tasks using the EC2 launch type require that the container instances that they're launched on to run at least version 1.25.1 of the Amazon ECS container agent. For more information, see Amazon ECS Container Agent Versions (p. 142).
- 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.
- 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 port 443 from the private subnet of the VPC.
- Controlling access to Amazon ECS by attaching an endpoint policy to the VPC endpoint isn't currently supported. By default, full access to the service will be allowed through the endpoint. For more information, see Controlling Access to Services with VPC Endpoints in the Amazon VPC User Guide.

Creating the VPC Endpoints for Amazon ECS

To create the VPC endpoint for the Amazon ECS service, use the Creating an Interface 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.

- 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.

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. For more information, see [Connect to Your Container Instance](p. 113).
2. Stop the container agent.

   ```bash
   sudo docker stop ecs-agent
   ```
3. Start the container agent.

   ```bash
   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.
Amazon ECR Repositories

Amazon ECR is a managed AWS Docker registry service. Customers can use the familiar Docker CLI to push, pull, and manage images. Amazon ECR provides a secure, scalable, and reliable registry. Amazon ECR supports private Docker repositories with resource-based permissions using AWS IAM so that specific users or Amazon EC2 instances can access repositories and images. Developers can use the Docker CLI to author and manage images.

For more information on how to create repositories, push and pull images from Amazon ECR, and set access controls on your repositories, see the Amazon Elastic Container Registry User Guide.

Using Amazon ECR Images with Amazon ECS

You can use your ECR images with Amazon ECS, but you need to satisfy some prerequisites:

- Your container instances must be using at least version 1.7.0 of the Amazon ECS container agent. The latest version of the Amazon ECS–optimized AMI supports ECR images in task definitions. For more information, including the latest Amazon ECS–optimized AMI IDs, see Amazon ECS Container Agent Versions (p. 142).
- The Amazon ECS container instance role (ecsInstanceRole) that you use with your container instances must possess 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 (p. 415).

- In your ECS task definitions, make sure that you are using the full registry/repository:tag naming for your ECR images. For example, `aws_account_id.dkr.ecr.region.amazonaws.com/my-web-app:latest`. 
Monitoring Amazon ECS

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.

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; however, you should 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 level.

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 and reservation utilization metrics for your Amazon ECS clusters
- The CPU and memory utilization metrics for your Amazon ECS services

Topics

- Monitoring Tools (p. 369)
- Amazon ECS CloudWatch Metrics (p. 371)
- Amazon ECS Event Stream for CloudWatch Events (p. 384)
- Logging Amazon ECS API Calls with AWS CloudTrail (p. 395)

Monitoring Tools

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 [Amazon ECS CloudWatch Metrics](p. 371).

  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 more information, see [Tutorial: Scaling Container Instances with CloudWatch Alarms](p. 379).

- **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. This is the only supported method for accessing logs for tasks using the Fargate launch type, but also works with tasks using the EC2 launch type. For more information, see [Using the awslogs Log Driver](p. 245).

  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. For more information, see [Using CloudWatch Logs with Container Instances](p. 114).

- **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 [Amazon ECS Event Stream for CloudWatch Events](p. 384) in this guide and Using Events in the [Amazon CloudWatch User Guide](API Version 2014-11-13 370).

- **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 [Logging Amazon ECS API Calls with AWS CloudTrail](p. 395) in this guide, and Working with CloudTrail Log Files in the [AWS CloudTrail User Guide](API Version 2014-11-13 370).

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.

- **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.
• 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.

Amazon ECS CloudWatch Metrics

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.

Topics
• Enabling CloudWatch Metrics (p. 371)
• Available Metrics and Dimensions (p. 371)
• Cluster Reservation (p. 374)
• Cluster Utilization (p. 376)
• Service Utilization (p. 377)
• Service RUNNING Task Count (p. 377)
• Viewing Amazon ECS Metrics (p. 378)
• Tutorial: Scaling Container Instances with CloudWatch Alarms (p. 379)

Enabling CloudWatch Metrics

Any task or service using the Fargate launch type is enabled for CloudWatch CPU and memory utilization metrics automatically, so you don’t need to take any manual steps.

For any task or service using the EC2 launch type, your Amazon ECS container instances require at least version 1.4.0 of the container agent to enable CloudWatch metrics, but 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 (p. 146).

If you're starting your agent manually (for example, if you're not using the Amazon ECS-optimized AMI for your container instances), see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-Optimized AMIs) (p. 150).

Your Amazon ECS container instances also require the ecs:StartTelemetrySession permission on the IAM role that you launch your container instances with. If you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, you might need to add this permission. For information about checking your Amazon ECS container instance role and attaching the managed IAM policy for container instances, see To check for the ecsInstanceRole in the IAM console (p. 417).

Note
You can disable CloudWatch metrics collection by setting ECS_DISABLE_METRICS=true in your Amazon ECS container agent configuration. For more information, see Amazon ECS Container Agent Configuration (p. 152).

Available Metrics and Dimensions

The following sections list the metrics and dimensions that Amazon ECS sends to Amazon CloudWatch.
Amazon Elastic Container Service Developer Guide
Available Metrics and Dimensions

Amazon ECS Metrics

Amazon ECS provides metrics for you to monitor your resources. You can measure the CPU and memory reservation and utilization across your cluster as a whole, and the CPU and memory utilization on the services in your clusters. For your GPU workloads, you can measure your GPU reservation across your cluster.

The metrics made available will depend on the launch type of the tasks and services in your clusters. If you're using the Fargate launch type for your services, CPU and memory utilization metrics are provided to assist in the monitoring of your services. For the EC2 launch type, you will own and need to monitor the Amazon EC2 instances that make your underlying infrastructure. Accordingly, additional CPU, memory, and GPU reservation and CPU and memory utilization metrics are made available at the cluster, service, and task level.

Amazon ECS sends the following metrics to CloudWatch every minute. When Amazon ECS collects metrics, it collects multiple data points every minute. It then aggregates them to one data point before sending the data to CloudWatch. So in CloudWatch, one sample count is actually the aggregate of multiple data points during one minute.

The AWS/ECS namespace includes the following metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUReservation</td>
<td>The percentage of CPU units that are reserved by running tasks in the cluster. Cluster CPU reservation (this metric can only be filtered by ClusterName) is measured as the total CPU units that are reserved by Amazon ECS tasks on the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster. This metric is only used for tasks using the EC2 launch type. Valid dimensions: ClusterName. Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average. Unit: Percent.</td>
</tr>
<tr>
<td>CPUUtilization</td>
<td>The percentage of CPU units that are used in the cluster or service. Cluster CPU utilization (metrics that are filtered by ClusterName without ServiceName) is measured as the total CPU units in use by Amazon ECS tasks on the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster. Cluster CPU utilization metrics are only used for tasks using the EC2 launch type. Service CPU utilization (metrics that are filtered by ClusterName and 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. Service CPU utilization metrics are used for tasks using both the Fargate and the EC2 launch type.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions: <code>ClusterName, ServiceName</code>.</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.</td>
</tr>
<tr>
<td></td>
<td>Unit: Percent.</td>
</tr>
<tr>
<td>MemoryReservation</td>
<td>The percentage of memory that is reserved by running tasks in the cluster.</td>
</tr>
<tr>
<td></td>
<td>Cluster memory reservation (this metric can only be filtered by <code>ClusterName</code>) is measured as the total memory that is reserved by Amazon ECS tasks on the cluster, divided by the total amount of memory that was registered for all of the container instances in the cluster. This metric is only used for tasks using the EC2 launch type.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions: <code>ClusterName</code>.</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.</td>
</tr>
<tr>
<td></td>
<td>Unit: Percent.</td>
</tr>
<tr>
<td>MemoryUtilization</td>
<td>The percentage of memory that is used in the cluster or service.</td>
</tr>
<tr>
<td></td>
<td>Cluster memory utilization (metrics that are filtered by <code>ClusterName</code> without <code>ServiceName</code>) is measured as the total memory in use by Amazon ECS tasks on the cluster, divided by the total amount of memory that was registered for all of the container instances in the cluster. Cluster memory utilization metrics are only used for tasks using the EC2 launch type.</td>
</tr>
<tr>
<td></td>
<td>Service memory utilization (metrics that are filtered by <code>ClusterName</code> and <code>ServiceName</code>) is measured as the total memory in use by the tasks that belong to the service, divided by the total memory that is reserved for the tasks that belong to the service. Service memory utilization metrics are used for tasks using both the Fargate and EC2 launch types.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions: <code>ClusterName, ServiceName</code>.</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.</td>
</tr>
<tr>
<td></td>
<td>Unit: Percent.</td>
</tr>
</tbody>
</table>
Cluster Reservation

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. This metric is used only on clusters with tasks or services using the EC2 launch type. It's not supported on clusters with tasks using the Fargate launch type.

\[
\text{Cluster CPU reservation} = \frac{\text{(Total CPU units reserved by tasks in cluster)} \times 100}{\text{(Total CPU units registered by container instances in cluster)}},
\]

### Metric Description

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPUReservation</td>
<td>The percentage of total available GPUs that are reserved by running tasks in the cluster. Cluster GPU reservation 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 GPU-enabled container instances in the cluster. Valid dimensions: ClusterName. Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average. Unit: Percent.</td>
</tr>
</tbody>
</table>

### Dimensions for Amazon ECS Metrics

Amazon ECS metrics use the `AWS/ECS` namespace and provide metrics for the following dimensions.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClusterName</td>
<td>This dimension filters the data that you request for all resources in a specified cluster. All Amazon ECS metrics are filtered by ClusterName.</td>
</tr>
<tr>
<td>ServiceName</td>
<td>This dimension filters the data that you request for all resources in a specified service within a specified cluster.</td>
</tr>
</tbody>
</table>

### Clusters

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. This metric is used only on clusters with tasks or services using the EC2 launch type. It's not supported on clusters with tasks using the Fargate launch type.
Cluster memory reservation =
\[
\frac{(\text{Total MiB of memory reserved by tasks in cluster} \times 100)}{\text{(Total MiB of memory registered by container instances in cluster)}}
\]

Cluster GPU reservation =
\[
\frac{(\text{Total GPUs reserved by tasks in cluster} \times 100)}{\text{(Total GPUs registered by container instances in cluster)}}
\]

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), 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 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 (p. 202) task definition parameter documentation.
Cluster Utilization

Cluster utilization is measured as the percentage of CPU and memory that is used by all Amazon ECS tasks on a cluster when compared to the aggregate CPU and memory that was registered for each active container instance in the cluster. A GPU utilization metric isn’t supported because it’s not possible to overcommit a GPU. This metric is used only on clusters with tasks or services using the EC2 launch type. It’s not supported on clusters with tasks using the Fargate launch type.

\[
\text{Cluster CPU utilization} = \frac{\text{(Total CPU units used by tasks in cluster)}}{\text{(Total CPU units registered by container instances in cluster)}} \times 100
\]

\[
\text{Cluster memory utilization} = \frac{\text{(Total MiB of memory used by tasks in cluster \times 100)}}{\text{(Total MiB of memory registered by container instances in cluster)}}
\]

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task that is running on that container 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 container instances registered, a \text{c4.4xlarge} instance and a \text{c4.large} instance. The \text{c4.4xlarge} instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The \text{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.

**Service Utilization**

Service 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. This metric is supported for services with tasks using both the EC2 and Fargate launch types.

\[
\text{Service CPU utilization} = \frac{(\text{Total CPU units used by tasks in service}) \times 100}{(\text{Total CPU units specified in task definition}) \times (\text{number of tasks in service})}
\]

\[
\text{Service memory utilization} = \frac{(\text{Total MiB of memory used by tasks in service}) \times 100}{(\text{Total MiB of memory specified in task definition}) \times (\text{number of tasks in service})}
\]

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task owned by the service that is running on that container instance, and 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. 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 Definition Parameters.

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 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.

**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.
To view the number of running tasks in a service

2. On the navigation pane, choose Metrics.
3. On the All metrics tab, choose ECS.
4. Choose ClusterName, ServiceName and then choose any metric (either CPUUtilization or MemoryUtilization) that corresponds to the service to view running tasks in.
5. On the Graphed metrics tab, change Period to 1 Minute and Statistic to Sample Count.

The value displayed in the graph indicates the number of RUNNING tasks in the service.

Viewing Amazon ECS Metrics

After you have enabled CloudWatch metrics for Amazon ECS, you can view those 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.

Topics

- Viewing Cluster Metrics on the Amazon ECS Console (p. 378)
- Viewing Service Metrics on the Amazon ECS Console (p. 379)
- Viewing Amazon ECS Metrics on the CloudWatch Console (p. 379)

Viewing Cluster Metrics on the Amazon ECS Console

Cluster and service metrics are available on the Amazon ECS console. The view provided for cluster 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 about cluster metrics, see Cluster Reservation (p. 374) and Cluster Utilization (p. 376).

To view cluster metrics on the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Select the cluster that you want to view metrics for.
Viewing Service Metrics on the 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 Service Utilization (p. 377).

To view service metrics in the console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Select the cluster that contains the service that you want to view metrics for.
4. Choose the service that you want to view metrics for.
5. On the Service: *service-name* page, choose Metrics.

Viewing Amazon ECS Metrics on the CloudWatch Console

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. You can view Cluster Reservation (p. 374), Cluster Utilization (p. 376), Service Utilization (p. 377), and the Service RUNNING Task Count (p. 377). For more information, see the Amazon CloudWatch User Guide.

To view metrics in the CloudWatch console

2. In the Metrics section in the navigation pane, choose ECS.
3. Choose the metrics to view. Cluster metrics are scoped as ECS > *ClusterName* and service utilization metrics are scoped as ECS > *ClusterName*, *ServiceName*. The following example shows cluster CPU and memory utilization.

![CPU and Memory Utilization Chart](chart.png)

Tutorial: Scaling Container Instances with CloudWatch Alarms

The following procedures help you to create an Auto Scaling group for an Amazon ECS cluster. The Auto Scaling group contains container instances that you can scale up (and down) using CloudWatch alarms.

Depending on the Amazon EC2 instance types that you use in your clusters, and quantity of container instances that you have in a cluster, your tasks have a limited amount of resources that they can use...
while running. Amazon ECS monitors the resources available in the cluster to work with the schedulers to
place tasks. If your cluster runs low on any of these resources, such as memory, you are eventually unable
to launch more tasks until you add more container instances, reduce the number of desired tasks in a
service, or stop some of the running tasks in your cluster to free up the constrained resource.

In this tutorial, you create a CloudWatch alarm using the MemoryReservation metric for your cluster.
When the memory reservation of your cluster rises above 75% (meaning that only 25% of the memory
in your cluster is available to for new tasks to reserve), the alarm triggers the Auto Scaling group to add
another instance and provide more resources for your tasks and services.

**Prerequisites**

This tutorial assumes that you have enabled CloudWatch metrics for your clusters and services. 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 yet.

Your Amazon ECS container instances require at least version 1.4.0 of the container agent to enable
CloudWatch metrics. For information about checking your agent version and updating to the latest
version, see Updating the Amazon ECS Container Agent (p. 146).

Your Amazon ECS container instances also require ecs:StartTelemetrySession permission on
the IAM role that you launch your container instances with. If you created your Amazon ECS container
instance role before CloudWatch metrics were available for Amazon ECS, then you might need to add
this permission. For information about checking your Amazon ECS container instance role and attaching
the managed IAM policy for container instances, see To check for the ecsInstanceRole in the IAM
console (p. 417).

**Step 1: Create a CloudWatch Alarm for a Metric**

After you have enabled CloudWatch metrics for your clusters and services, and the metrics for your
cluster are visible in the CloudWatch console, you can set alarms on the metrics. For more information,
see Creating Amazon CloudWatch Alarms in the Amazon CloudWatch User Guide.

For this tutorial, you create an alarm on the cluster MemoryReservation metric to alert when the
cluster's memory reservation is above 75%.

**To create a CloudWatch alarm on a metric**

2. On the left navigation, choose Alarms, Create Alarm.
3. In the CloudWatch Metrics by Category section, choose ECS Metrics > ClusterName.
4. On the Modify Alarm page, choose the MemoryReservation metric for the default cluster and
choose Next.
5. In the Alarm Threshold section, enter a name and description for your alarm.
   - **Name:** memory-above-75-pct
   - **Description:** Cluster memory reservation above 75%
6. Set the threshold and time period requirement to MemoryReservation greater than 75% for 1
   period.
7. (Optional) Configure a notification to send when the alarm is triggered. You can also choose to delete the notification if you don’t want to configure one now.

8. Choose Create Alarm. Now you can use this alarm to trigger your Auto Scaling group to add a container instance when the memory reservation is above 75%.

9. (Optional) You can also create another alarm that triggers when the memory reservation is below 25%, which you can use to remove a container instance from your Auto Scaling group.

### Step 2: Create a Launch Configuration for an Auto Scaling Group

Now that you have enabled CloudWatch metrics and created an alarm based on one of those metrics, you can create a launch configuration and an Auto Scaling group for your cluster. For more information and other configuration options, see the Amazon EC2 Auto Scaling User Guide.

**To create an Auto Scaling launch configuration**

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](https://console.aws.amazon.com/ec2/).
2. On the left navigation pane, choose **Auto Scaling Groups**.
3. On the **Welcome to Auto Scaling** page, choose **Create Auto Scaling Group**.
4. On the **Create Auto Scaling Group** page, choose **Create a new launch configuration**.
5. On the **Choose AMI** step of the **Create Auto Scaling Group** wizard, choose **Community AMIs**.
6. Choose the ECS-optimized AMI for your Auto Scaling group.

   To use the Amazon ECS-optimized Amazon Linux 2 AMI, type `amzn2-ami-ecs` in the **Search community AMIs** field and press the Enter key. Choose Select next to the `amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs` AMI.

The following table lists the current Amazon ECS-optimized Amazon Linux 2 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-012ca23958772cf77</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-00cf4737e2388664b9</td>
<td>Launch instance</td>
</tr>
<tr>
<td>Region</td>
<td>AMI Name</td>
<td>AMI ID</td>
<td>EC2 Console Link</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------</td>
<td>-----------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>us-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-06d87f0156b1d440</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0a9f5be2a016dceca</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-067f4f7124e746e8b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0e52aad6ac7733af6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-08834c8c57e502da6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-04322e867758d97a7</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0047bfdb16f1f67b8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0947584732e556c6a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-049ac464ec4d2ba3d</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-096a38c97b80cd8e5</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0dddc4daca44e6eb6</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0c5ab4df76eaba4b</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-0204aa6a92a5456e1</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn2-ami-ecs-hvm-2.0.20190510-x86_64-ebs</td>
<td>ami-07273195833e4f20d</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>
Region | AMI Name | AMI ID | EC2 Console Link
--- | --- | --- | ---
sa-east-1 | `ami2-ami-ecs-hvm-2.0.20190510-x86_64-ebs` | `ami-00d851648873aaabc` | Launch instance
us-gov-east-1 | `ami2-ami-ecs-hvm-2.0.20190510-x86_64-ebs` | `ami-06473be43b0f77600` | Launch instance
us-gov-west-1 | `ami2-ami-ecs-hvm-2.0.20190510-x86_64-ebs` | `ami-607c0001` | Launch instance

7. On the **Choose Instance Type** step of the Create Auto Scaling Group wizard, choose an instance type for your Auto Scaling group and choose Next: Configure details.

8. On the **Configure details** step of the Create Auto Scaling Group wizard, enter the following information. The other fields are optional. For more information, see Creating Launch Configurations in the Amazon EC2 Auto Scaling User Guide.

   - **Name**: Enter a name for your launch configuration.
   - **IAM role**: Select the `ecsInstanceRole` for your container instances. If you do not have this role configured, see Amazon ECS Container Instance IAM Role (p. 415).
   - **IP Address Type**: Select the IP address type option for your container instances. To allow external traffic to be able to reach your containers, choose Assign a public IP address to every instance.

9. (Optional) If you have configuration information to pass to your container instances with Amazon EC2 user data, choose Advanced Details and enter your user data in the User data field. For more information, see Amazon ECS Container Agent Configuration (p. 152).


12. On the Configure Security Group step of the Create Auto Scaling Group wizard, select an existing security group that meets the needs of your containers, or create a new security group, and choose Review.

13. Review your launch configuration and choose Create launch configuration.

14. Select a private key to use for connecting to your instances with SSH and choose Create launch configuration. Move on to creating an Auto Scaling group with your new launch configuration.

### Step 3: Create an Auto Scaling Group for your Cluster

After the launch configuration is complete, continue with the following procedure to create an Auto Scaling group that uses your launch configuration.

#### To create an Auto Scaling group

1. On the Configure Auto Scaling group details step of the Create Auto Scaling Group wizard, enter the following information and then choose Next: Configure scaling policies:

   - **Group name**: Enter a name for your Auto Scaling group.
   - **Group size**: Specify the number of container instances with which your Auto Scaling group should start.
   - **Network**: Select a VPC into which to launch your container instances.
   - **Subnet**: Select the subnets into which to launch your container instances. For a highly available cluster, we recommend that you enable all of the subnets in the Region.

---

API Version 2014-11-13
383
2. On the **Configure scaling policies** step of the **Create Auto Scaling Group** wizard, choose **Use scaling policies to adjust the capacity of this group**.

3. Enter the minimum and maximum number of container instances for your Auto Scaling group.

4. In the **Increase Group Size** section, enter the following information:
   - **Execute policy when**: Select the **memory-above-75-pct** CloudWatch alarm that you configured earlier.
   - **Take the action**: Enter the number of instances to add to your cluster when the alarm is triggered.

5. If you configured an alarm to trigger a group size reduction, set that alarm in the **Decrease Group Size** section and specify how many instances to remove if that alarm is triggered. Otherwise, collapse the **Decrease Group Size** section by choosing the X in the upper-right-hand corner of the section.

   **Note**
   If you configure your Auto Scaling group to remove container instances, any tasks running on the removed container instances are stopped. If your tasks are running as part of a service, Amazon ECS restarts those tasks on another instance if the required resources are available (CPU, memory, ports). However, tasks that were started manually are not restarted automatically.

6. Choose **Review, Create Auto Scaling Group**.

**Step 4: Verify and Test your Auto Scaling Group**

Now that you've created your Auto Scaling group, you should be able to see your instances launching in the Amazon EC2 console **Instances** page. These instances should register into your Amazon ECS cluster as well after they launch.

To test that your Auto Scaling group is configured properly, create some tasks that consume a considerable amount of memory and start launching them into your cluster. After your cluster exceeds the 75% memory reservation from the CloudWatch alarm for the specified number of periods, you should see a new instance launch in the Amazon EC2 console.

**Step 5: Cleaning Up**

When you have completed this tutorial, you may choose to keep your Auto Scaling group and Amazon EC2 instances in service for your cluster. However, if you are not actively using these resources, you should consider cleaning them up so your account does not incur unnecessary charges. You can delete your Auto Scaling group to terminate the Amazon EC2 instances within it, but your launch configuration remains intact. You can create a new Auto Scaling group with the launch configuration later, if you choose.

**To delete your Auto Scaling group**

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](https://console.aws.amazon.com/ec2/).
2. On the left navigation pane, choose **Auto Scaling Groups**.
3. Choose the Auto Scaling group that you created earlier.
4. Choose **Actions, Delete**.
5. Choose **Yes, Delete**.

**Amazon ECS Event Stream for CloudWatch Events**

You can use Amazon ECS event stream for CloudWatch Events to receive near real-time notifications regarding the current state of your Amazon ECS clusters. If your tasks are using the Fargate launch type,
you can see the state of your tasks. 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.

Using CloudWatch Events, you can build custom schedulers on top of Amazon ECS that are responsible for orchestrating tasks across clusters, and to monitor 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 CloudWatch Events target, such as AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, and Amazon Kinesis Data Streams.

An Amazon ECS event stream ensures that every event delivered at least one time. If duplicate events are sent, the event provides enough information to identify duplicates. For more information, see Handling Events (p. 390).

Events are relatively ordered, so that you can easily tell when an event occurred in relation to other events.

Topics
- Amazon ECS Events (p. 385)
- Handling Events (p. 390)
- Tutorial: Listening for Amazon ECS CloudWatch Events (p. 391)
- Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events (p. 393)

Amazon ECS Events

Amazon ECS sends two types of events to CloudWatch Events: container instance events and task events. Container instance events are only sent if you are using the EC2 launch type for our tasks. For tasks using the Fargate launch type you only receive task state events. Amazon ECS tracks the state of container instances and tasks. If either resources changes, an event is triggered. These events are classified as either container instance state change events or task state change events. These events and their possible causes are described in greater detail in the following sections.

Note
Amazon ECS may add other event types, sources, and details in the future. If you are programmatically deserializing event JSON data, 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 triggered for the same activity. For example, when a task is started on a container instance, a task state change event is triggered for the new task. A container instance state change event is triggered 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 triggered for the container instance, the container agent connection status, and every task that was running on the container instance.

Events contain two version fields; one in the main body of the event, and one in the detail object of the event.

- The version in the main body of the event is set to 0 on all events. For more information about CloudWatch Events parameters, see Events and Event Patterns in the Amazon CloudWatch Events User Guide.
- The version 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 (they have the same version in the detail object). If you are replicating your Amazon ECS container instance and task state with CloudWatch Events, you can compare the version of a resource reported by the Amazon ECS APIs with the version reported in CloudWatch Events for the resource (inside the detail object) to verify that the version in your event stream is current.
Topics

- Container Instance State Change Events (p. 386)
- Task State Change Events (p. 389)

Container Instance State Change Events

The following scenarios trigger 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 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 periodically disconnects and reconnects (several times per hour) as a part of its normal operation, so agent connection events should be expected and they 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 triggers 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 object that is returned from a DescribeContainerInstances API operation).
API operation in the Amazon Elastic Container Service API Reference. For more information about CloudWatch Events parameters, see Events and Event Patterns in the Amazon CloudWatch Events 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"
      }
    ],
    "containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/b54a2a04-046f-4331-9d74-3f6d7f6ca315"
  }
```

API Version 2014-11-13
387
"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"}
Task State Change Events

The following scenarios trigger 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.
- The Amazon ECS container agent monitors the state of tasks on your container instances, and it reports any state changes (for example, 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 triggers an event.

Example Task State Change Event

Task state change events are delivered in the following format (the `detail` section below resembles the `Task` object that is returned from a `DescribeTasks` API operation in the Amazon Elastic Container Service API Reference). For more information about CloudWatch Events parameters, see Events and Event Patterns in the Amazon CloudWatch Events User Guide.

```json
{
  "version": "0",
  "id": "9bcdac79-b31f-4d3d-9410-fbd727c29fab",
  "detail-type": "ECS Task 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:task/b99d40b3-5176-4f71-9a52-9dbd6f1cebef"],
  "detail": {
    "containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/b54a2a04-046f-4331-9d74-5f6d7f6ca315",
    "containers": [
      {
        
```
Handling Events

Amazon ECS sends events on an "at least once" basis. This means you may receive more than a single copy of a given event. Additionally, events may not be delivered to your event listeners in the order in which the events occurred.

To enable proper ordering of events, the detail section of each event contains a version property. Events with a higher version property number should be treated as occurring later than events with lower version numbers. Events with matching version numbers can be treated as duplicates.

Example: Handling Events in an AWS Lambda Function

The following example shows a Lambda function written in Python 2.7 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.

```python
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
    )

Tutorial: Listening for Amazon ECS CloudWatch Events

In this tutorial, you set up a simple AWS Lambda function that listens for Amazon ECS 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 Getting Started with Amazon ECS (p. 19) 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 Python 2.7.
   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:

   ```python
   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 2.7 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 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 and Event Patterns in the Amazon CloudWatch Events User Guide.

To route events to your Lambda function

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: 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 Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose Clusters, default.
4. For Task Definition, select the latest version of console-sample-app-static and choose Run Task.
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.

Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events

In this tutorial, you configure a CloudWatch Events 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 Amazon ECS (p. 19) 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 CloudWatch Events event rule correctly.

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.

To create an Amazon SNS topic

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v2/home.
2. Choose Topics, Create new topic.
3. On the Create new topic window, for Topic name, enter TaskStoppedAlert and choose Create topic.
4. On the Topics window, select the topic that you just created. On the Topic details: TaskStoppedAlert screen, choose Create subscription.

5. On the Create Subscription window, for Protocol, choose Email. For Endpoint, enter an email address to which you currently have access and choose Create subscription.

6. Check your email account, and wait to receive a subscription confirmation email message. When you receive it, choose Confirm subscription.

Step 2: Register Event Rule

Next, you register an event rule that captures only task-stopped events for tasks with stopped containers.

To create an event rule

2. On the navigation pane, choose Events, Create rule.
3. Choose Show advanced options, edit.
4. For Build a pattern that selects events for processing by your targets, replace the existing text with the following text:

```
{
  "source": [  
    "aws.ecs"
  ],
  "detail-type": [  
    "ECS Task State Change"
  ],
  "detail": {  
    "lastStatus": [  
      "STOPPED"
    ],
    "stoppedReason": [  
      "Essential container in task exited"
    ]
  }
}
```

This code defines a CloudWatch Events event rule that matches any event where the lastStatus and stoppedReason fields match the indicated values. For more information about event patterns, see Events and Event Patterns in the Amazon CloudWatch User Guide.

5. For Targets, choose Add target. For Target type, choose SNS topic, and then choose TaskStoppedAlert.

6. Choose Configure details.

7. For Rule definition, type a name and description for your rule and then choose Create rule.

Step 3: Test Your Rule

To test your rule, you attempt to run 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.

To test a rule

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose Task Definitions, Create new Task Definition.
3. For Task Definition Name, type WordPressFailure and choose Add Container.
4. For Container name, type Wordpress, for Image, type wordpress, and for Maximum memory (MB), type 128.
5. Choose Add, Create.
7. For Cluster, choose default. Choose Run Task.
8. On the Tasks tab for your cluster, periodically choose the refresh icon until you no longer see your task running. To verify that your task has stopped, for Desired task status, choose Stopped.
9. Check your email to confirm that you have received an email alert for the stopped notification.

Logging Amazon ECS API Calls with AWS CloudTrail

Amazon ECS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon ECS. CloudTrail captures all API calls for Amazon ECS as events, including calls from the Amazon ECS console and from code calls to the Amazon ECS API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon ECS. If you don’t configure a trail, you can still view the most recent events in the CloudTrail console in Event history. 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 the AWS CloudTrail User Guide.

Amazon ECS Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When 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. A trail enables CloudTrail 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:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon ECS actions are logged by CloudTrail and are documented in the Amazon Elastic Container Service API Reference. For example, calls to the CreateService, RunTask and DeleteCluster sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:
For more information, see the CloudTrail userIdentity Element.

Understanding Amazon ECS Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

Note
These examples have 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.

The following example shows a CloudTrail log entry that demonstrates the CreateCluster action:

```json
{
    "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"
}
Amazon ECS IAM Policies, Roles, and Permissions

By default, IAM users don't have permission to create or modify Amazon ECS resources, or perform tasks using the Amazon ECS API. This means that they also can't do so using the Amazon ECS console or the AWS CLI. To allow IAM users to create or modify resources and perform tasks, you must create IAM policies. Policies grant IAM users permissions to use specific resources and API actions. Then, attach those policies to the IAM users or groups that require those permissions.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more general information about IAM policies, see Permissions and Policies in the IAM User Guide. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

Likewise, Amazon ECS container instances make calls to the Amazon ECS and Amazon EC2 APIs on your behalf, so they need to authenticate with your credentials. This authentication is accomplished by creating an IAM role for your container instances and associating that role with your container instances when you launch them. For more information, see Amazon ECS Container Instance IAM Role (p. 415). If you use an Elastic Load Balancing load balancer with your Amazon ECS services, calls to the Amazon EC2 and Elastic Load Balancing APIs are made on your behalf to register and deregister container instances with your load balancers. For more information, see Amazon ECS Service Scheduler IAM Role (p. 426). For more general information about IAM roles, see IAM Roles in the IAM User Guide.

Getting Started

An IAM policy must grant or deny permission to use one or more Amazon ECS actions. It must also specify the resources that can be used with the action, which can be all resources, or in some cases, specific resources. The policy can also include conditions that you apply to the resource.

Amazon ECS partially supports resource-level permissions. This means that for some Amazon ECS API actions, you cannot specify which resource a user is allowed to work with for that action; instead, you have to allow users to work with all resources for that action.

Topics

- Policy Structure (p. 399)
- Supported Resource-Level Permissions for Amazon ECS API Actions (p. 402)
- Creating Amazon ECS IAM Policies (p. 407)
- Managed Policies and Trust Relationships (p. 407)
- Amazon ECS Container Instance IAM Role (p. 415)
- Amazon ECS Task Execution IAM Role (p. 418)
- Using Service-Linked Roles for Amazon ECS (p. 421)
- Amazon ECS Service Scheduler IAM Role (p. 426)
- Amazon ECS CodeDeploy IAM Role (p. 428)
- Amazon ECS Service Auto Scaling IAM Role (p. 431)
- Amazon ECS Task Role (p. 432)
- CloudWatch Events IAM Role (p. 433)
- IAM Roles for Tasks (p. 435)
Policy Structure

The following topics explain the structure of an IAM policy.

Topics

- Policy Syntax (p. 399)
- Actions for Amazon ECS (p. 400)
- Amazon Resource Names for Amazon ECS (p. 400)
- Condition Keys for Amazon ECS (p. 401)
- Checking that Users Have the Required Permissions (p. 402)

Policy Syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows:

```json
{
    "Statement": [{
        "Effect": "effect",
        "Action": "action",
        "Resource": "arn",
        "Condition": {
            "key": "value"
        }
    }
}
```

There are various elements that make up a statement:

- **Effect**: The `effect` can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.

- **Action**: The `action` is the specific API action for which you are granting or denying permission. To learn about specifying `action`, see Actions for Amazon ECS (p. 400).

- **Resource**: The resource that's affected by the action. Some Amazon ECS API actions allow you to include specific resources in your policy that can be created or modified by the action. To specify a resource in the statement, use its Amazon Resource Name (ARN). For more information about specifying the `arn` value, see Amazon Resource Names for Amazon ECS (p. 400). For more information about which API actions support which ARNs, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 402). If the API action does not support ARNs, use the * wildcard to specify that all resources can be affected by the action.

- **Condition**: Conditions are optional. They can be used to control when your policy is in effect. For more information about specifying conditions for Amazon ECS, see Condition Keys for Amazon ECS (p. 401).

For more information about example IAM policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 407).
Actions for Amazon ECS

In an IAM policy statement, you can specify any API action from any service that supports IAM. For Amazon ECS, use the following prefix with the name of the API action: `ecs:`. For example: `ecs:RunTask` and `ecs:CreateCluster`.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["ecs:action1", "ecs:action2"]
```

You can also specify multiple actions using wildcards. For example, you can specify all actions whose name begins with the word "Describe" as follows:

```
"Action": "ecs:Describe*"
```

To specify all Amazon ECS API actions, use the * wildcard as follows:

```
"Action": "ecs:*"
```

For a list of Amazon ECS actions, see Actions in the Amazon Elastic Container Service API Reference.

Amazon Resource Names for Amazon ECS

Each IAM policy statement applies to the resources that you specify using their ARNs.

**Important**
Currently, not all API actions support individual ARNs. For information about which ARNs you can use with which Amazon ECS API actions, as well as supported condition keys for each ARN, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 402).

An ARN has the following general syntax:

```
arn:aws:[service]:[region]:[account]:resourceType/resourcePath
```

- **service**
  The service (for example, `ecs`).

- **region**
  The Region for the resource (for example, `us-east-1`).

- **account**
  The AWS account ID, with no hyphens (for example, `123456789012`).

- **resourceType**
  The type of resource (for example, `instance`).

- **resourcePath**
  A path that identifies the resource. You can use the * wildcard in your paths.

For example, you can indicate a specific cluster (default) in your statement using its ARN as follows:

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/default"
```
You can also specify all clusters that belong to a specific account by using the * wildcard as follows:

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

To specify all resources, or if a specific API action does not support ARNs, use the * wildcard in the Resource element as follows:

```
"Resource": "*"
```

The following table describes the ARNs for each type of resource used by the Amazon ECS API actions.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Amazon ECS resources</td>
<td>arn:aws:ecs:*</td>
</tr>
<tr>
<td>All Amazon ECS resources</td>
<td>arn:aws:ecs:region:account:*</td>
</tr>
<tr>
<td>owned by the specified account in the specified region</td>
<td>arn:aws:ecs:region:account:cluster/cluster-name</td>
</tr>
<tr>
<td>Task definition</td>
<td>arn:aws:ecs:region:account:task-definition/task-definition-family-name:task-definition-revision-number</td>
</tr>
<tr>
<td>Service</td>
<td>arn:aws:ecs:region:account:service/service-name</td>
</tr>
<tr>
<td>Task</td>
<td>arn:aws:ecs:region:account:task/task-id</td>
</tr>
<tr>
<td>Container</td>
<td>arn:aws:ecs:region:account:container/container-id</td>
</tr>
</tbody>
</table>

Many Amazon ECS API actions accept multiple resources. To specify multiple resources in a single statement, separate their ARNs with commas, as follows:

```
"Resource": ["arn1", "arn2"]
```

For more general information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the Amazon Web Services General Reference.

### Condition Keys for Amazon ECS

In a policy statement, you can optionally specify conditions that control when it is in effect. Each condition contains one or more key-value pairs. Condition keys are not case-sensitive. We've defined AWS-wide condition keys, plus additional service-specific condition keys.

If you specify multiple conditions, or multiple keys in a single condition, we evaluate them using a logical AND operation. If you specify a single condition with multiple values for one key, we evaluate the condition using a logical OR operation. For permission to be granted, all conditions must be met.

You can also use placeholders when you specify conditions. For more information, see Policy Variables in the IAM User Guide.

Amazon ECS implements the AWS-wide condition keys (see Available Keys), plus the following service-specific condition keys.
<table>
<thead>
<tr>
<th>Condition Key</th>
<th>Key/Value Pair</th>
<th>Evaluation Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecs:cluster</td>
<td>&quot;ecs:cluster&quot;:&quot;cluster-arn&quot;</td>
<td>ARN, Null</td>
</tr>
<tr>
<td></td>
<td>Where <code>cluster-arn</code> is the ARN for the Amazon ECS cluster</td>
<td></td>
</tr>
<tr>
<td>ecs:container-instances</td>
<td>&quot;ecs:container-instances&quot;:&quot;container-instance-arns&quot;</td>
<td>ARN, Null</td>
</tr>
<tr>
<td></td>
<td>Where <code>container-instance-arns</code> is one or more container instance ARNs.</td>
<td></td>
</tr>
</tbody>
</table>

For information about which condition keys you can use with which Amazon ECS resources, on an action-by-action basis, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 402). For example policy statements for Amazon ECS, see Creating Amazon ECS IAM Policies (p. 407).

## Checking that Users Have the Required Permissions

After you've created an IAM policy, we recommend that you check whether it grants users the permissions to use the particular API actions and resources they need before you put the policy into production.

First, create an IAM user for testing purposes, and then attach the IAM policy that you created to the test user. Then, make a request as the test user. You can make test requests in the console or with the AWS CLI.

**Note**

You can also test your policies with the IAM Policy Simulator. For more information on the policy simulator, see Working with the IAM Policy Simulator in the IAM User Guide.

If the action that you are testing creates or modifies a resource, you should make the request using the `DryRun` parameter (or run the AWS CLI command with the `--dry-run` option). In this case, the call completes the authorization check, but does not complete the operation. For example, you can check whether the user can terminate a particular instance without actually terminating it. If the test user has the required permissions, the request returns `DryRunOperation`; otherwise, it returns `UnauthorizedOperation`.

If the policy doesn't grant the user the permissions that you expected, or is overly permissive, you can adjust the policy as needed and retest until you get the desired results.

**Important**

It can take several minutes for policy changes to propagate before they take effect. Therefore, we recommend that you allow five minutes to pass before you test your policy updates.

If an authorization check fails, the request returns an encoded message with diagnostic information. You can decode the message using the `DecodeAuthorizationMessage` action. For more information, see `DecodeAuthorizationMessage` in the AWS Security Token Service API Reference, and `decode-authorization-message` in the AWS CLI Command Reference.

## Supported Resource-Level Permissions for Amazon ECS API Actions

The term **Resource-level permissions** refers to the ability to specify which resources users are allowed to perform actions on. Amazon ECS has partial support for resource-level permissions. This means that
for certain Amazon ECS actions, you can control when users are allowed to use those actions based on conditions that have to be fulfilled, or specific resources that users are allowed to use. For example, you can grant users permission to launch instances, but only of a specific type, and only using a specific AMI.

The following table describes the Amazon ECS API actions that currently support resource-level permissions, as well as the supported resources, resource ARNs, and condition keys for each action.

Important
If an Amazon ECS API action in this table is marked N/A for Resource Types, then it does not support resource-level permissions. If an Amazon ECS API action does not support resource-level permissions, you can grant users permission to use the action, but you have to specify a * for the resource element of your policy statement.

<table>
<thead>
<tr>
<th>Actions</th>
<th>Resource Types</th>
<th>Condition Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCluster</td>
<td>N/A</td>
<td>aws:RequestTag</td>
</tr>
<tr>
<td>CreateService</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CreateTaskSet</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DeleteAccountSettings</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DeleteAttributes</td>
<td>Container instance (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td>DeleteCluster</td>
<td>Cluster (required)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>arn:aws:ecs:region:account:cluster/my-cluster</td>
<td></td>
</tr>
<tr>
<td>DeleteService</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DeleteTaskSet</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DeregisterContainerInstance (required)</td>
<td>arn:aws:ecs:region:account:cluster/my-cluster</td>
<td></td>
</tr>
<tr>
<td>DeregisterTaskDefinition</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DescribeClusters</td>
<td>Cluster (required)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>arn:aws:ecs:region:account:cluster/my-cluster</td>
<td></td>
</tr>
<tr>
<td>DescribeContainerInstance</td>
<td>Container instance (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td>DescribeServices</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DescribeTaskDefinition</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Actions Supported Resource-Level Permissions

<table>
<thead>
<tr>
<th>Actions</th>
<th>Resource Types</th>
<th>Condition Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>DescribeTasks</td>
<td>Task (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td></td>
<td>[arn:aws:ecs:region:account:task/1abf0f6d-a411-4033-b8eb-a4eed3ad252a, arn:aws:ecs:region:account:task/1abf0f6d-a411-4033-b8eb-a4eed3ad252b]</td>
<td></td>
</tr>
<tr>
<td>DescribeTaskSets</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DiscoverPollEndpoint</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ListAccountSettings</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ListAttributes</td>
<td>Cluster (required)</td>
<td>N/A</td>
</tr>
<tr>
<td>ListClusters</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ListContainerInstances</td>
<td>Cluster (required)</td>
<td>N/A</td>
</tr>
<tr>
<td>ListServices</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ListTagsForResource</td>
<td>Cluster</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Container instance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[arn:aws:ecs:region:account:task/1abf0f6d-a411-4033-b8eb-a4eed3ad252a]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task definition</td>
<td></td>
</tr>
<tr>
<td>ListTaskDefinitionFamilies</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ListTaskDefinitions</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ListTasks</td>
<td>Container instance (required)</td>
<td>ecs:cluster</td>
</tr>
</tbody>
</table>
## Supported Resource-Level Permissions

<table>
<thead>
<tr>
<th>Actions</th>
<th>Resource Types</th>
<th>Condition Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poll</td>
<td>Container instance (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td>PutAccountSetting</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PutAccountSettingDefault</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PutAttributes</td>
<td>Container instance (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td>RegisterContainerInstance</td>
<td>Cluster (required)</td>
<td>aws:RequestTag</td>
</tr>
<tr>
<td></td>
<td>arn:aws:ecs:region:account:cluster/my-cluster</td>
<td></td>
</tr>
<tr>
<td>RegisterTaskDefinition</td>
<td>N/A</td>
<td>aws:RequestTag</td>
</tr>
<tr>
<td>RunTask</td>
<td>Task definition (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td></td>
<td>arn:aws:ecs:region:account:task-definition/hello_world:8</td>
<td>aws:RequestTag</td>
</tr>
<tr>
<td>StartTask</td>
<td>Task definition (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>aws:RequestTag</td>
</tr>
<tr>
<td>StartTelemetrySession</td>
<td>Container instance</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td>StopTask</td>
<td>Task (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td></td>
<td>arn:aws:ecs:region:account:task/1abf0f6d-a411-4033-b8eb-a4eed3ad252a</td>
<td></td>
</tr>
<tr>
<td>SubmitContainerStateChange</td>
<td>Cluster (required)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>arn:aws:ecs:region:account:cluster/my-cluster</td>
<td></td>
</tr>
<tr>
<td>SubmitTaskStateChange</td>
<td>Cluster (required)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>arn:aws:ecs:region:account:cluster/my-cluster</td>
<td></td>
</tr>
<tr>
<td>Actions</td>
<td>Resource Types</td>
<td>Condition Keys</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>TagResource</td>
<td>Cluster</td>
<td>aws:RequestTag</td>
</tr>
<tr>
<td></td>
<td>Container instance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task definition</td>
<td></td>
</tr>
<tr>
<td>UntagResource</td>
<td>Cluster</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Container instance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task definition</td>
<td></td>
</tr>
<tr>
<td>UpdateContainerAgent</td>
<td>Container instance (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UpdateContainerInstance</td>
<td>Container instance (required)</td>
<td>ecs:cluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UpdateService</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>UpdateSServicePrimaryTaskSet</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>UpdateTaskSet</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Creating Amazon ECS IAM Policies

You can create specific IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more information about IAM policies, see Permissions and Policies in the IAM User Guide. For more information about custom IAM policies, see Managing IAM Policies.

To create an IAM policy for a user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Policies, Create policy.
4. Choose Select actions and select the actions to add to the policy. For more information, see Amazon ECS IAM Policy Examples (p. 439).
5. (Optional) Choose Specify request conditions (optional) to add conditions to the policy that you are creating. Conditions limit a JSON policy statement's effect. For example, you can specify that a user is allowed to perform the actions on the resources only when that user's request happens within a certain time range. You can also use commonly used conditions to limit whether a user must be authenticated using a multi-factor authentication (MFA) device, or if the request must originate from within a certain range of IP addresses. For lists of all of the context keys that you can use in a policy condition, see AWS Service Actions and Condition Context Keys for Use in IAM Policies.
7. In the Name field, type your own unique name, such as AmazonECSUserPolicy.
8. Choose Create Policy to finish.

To attach an IAM policy to a user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Users and then choose the user you would like to attach the policy to.
3. Choose Permissions, Add permissions.
4. In the Grant permissions section, choose Attach existing policies directly.
5. Select the custom policy that you created in the previous procedure and choose Next: Review.
6. Review your details and choose Add permissions.

Managed Policies and Trust Relationships

Amazon ECS and Amazon ECR provide several managed policies and trust relationships that you can attach to IAM users, 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.

Topics
- Amazon ECS Managed Policies and Trust Relationships (p. 408)
- Amazon ECR Managed Policies (p. 414)
Amazon ECS Managed Policies and Trust Relationships

Amazon ECS provides several managed policies and trust relationships that you can attach to IAM users, EC2 instances, or Amazon ECS tasks that allow differing levels of control over Amazon ECS 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 each API operation mentioned in these policies, see Actions in the Amazon Elastic Container Service API Reference.

Topics
- AmazonECS_FullAccess (p. 408)
- AmazonEC2ContainerServiceFullAccess (p. 411)
- AmazonEC2ContainerServiceforEC2Role (p. 412)
- AmazonEC2ContainerServiceRole (p. 412)
- AmazonEC2ContainerServiceAutoscaleRole (p. 413)
- AmazonEC2ContainerServiceTaskRole (p. 413)
- AmazonEC2ContainerServiceEventsRole (p. 413)

AmazonECS_FullAccess

This managed policy provides administrative access to Amazon ECS resources and enables ECS features through access to other AWS service resources, including VPCs, Auto Scaling groups, and AWS CloudFormation stacks.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:Describe*",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:PutMetricAlarm",
                "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",
"lambda:ListFunctions",
"ec2:AssociateRouteTable",
"ec2:AttachInternetGateway",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:CancelSpotFleetRequests",
"ec2:CreateInternetGateway",
"ec2:CreateLaunchTemplate",
"ec2:CreateRoute",
"ec2:CreateRouteTable",
"ec2:CreateSecurityGroup",
"ec2:CreateSubnet",
"ec2:CreateVpc",
"ec2:DeleteLaunchTemplate",
"ec2:DeleteSubnet",
"ec2:DeleteVpc",
"ec2:Describe*",
"ec2:DetachInternetGateway",
"ec2:DisassociateRouteTable",
"ec2:ModifySubnetAttribute",
"ec2:ModifyVpcAttribute",
"ec2:RunInstances",
"ec2:RequestSpotFleet",
"elasticloadbalancing:CreateListener",
"elasticloadbalancing:CreateLoadBalancer",
"elasticloadbalancing:CreateRule",
"elasticloadbalancing:CreateTargetGroup",
"elasticloadbalancing:DeleteListener",
"elasticloadbalancing:DeleteLoadBalancer",
"elasticloadbalancing:DeleteRule",
"elasticloadbalancing:DeleteTargetGroup",
"elasticloadbalancing:DescribeLoadBalancers",
"elasticloadbalancing:DescribeTargetGroups",
"ecs:*",
"events:DescribeRule",
"events:DeleteRule",
"events:ListRuleNamesByTarget",
"events:ListTargetsByRule",
"events:PutRule",
"events:PutTargets",
"events:RemoveTargets",
"iam:ListAttachedRolePolicies",
"iam:ListInstanceProfiles",
"iam:ListRoles",
"logs:CreateLogGroup",
"logs:DescribeLogGroups",
"logs:FilterLogEvents",
"route53:GetHostedZone",
Amazon Elastic Container Service Developer Guide
Amazon ECS Managed Policies and Trust Relationships

"route53:ListHostedZonesByName",
"route53:CreateHostedZone",
"route53:DeleteHostedZone",
"route53:GetHealthCheck",
"servicediscovery:CreatePrivateDnsNamespace",
"servicediscovery:CreateService",
"servicediscovery:GetNamespace",
"servicediscovery:GetOperation",
"servicediscovery:GetService",
"servicediscovery:ListNamespaces",
"servicediscovery:ListServices",
"servicediscovery:UpdateService"
],
"Resource": ["*"]
},
{
"Effect": "Allow",
"Action": [
"ssm:GetParametersByPath",
"ssm:GetParameters",
"ssm:GetParameter"
],
"Resource": "arn:aws:ssm:*:*:parameter/aws/service/ecs*"
},
{
"Effect": "Allow",
"Action": [
"ec2:DeleteInternetGateway",
"ec2:DeleteRoute",
"ec2:DeleteRouteTable",
"ec2:DeleteSecurityGroup"
],
"Resource": ["*"]
},
"Condition": {
"StringLike": {
"ec2:ResourceTag/aws:cloudformation:stack-name": "EC2ContainerService-
*"
}
}
},
{
"Action": "iam:PassRole",
"Effect": "Allow",
"Resource": ["*"]
},
"Condition": {
"StringLike": {
"iam:PassedToService": "ecs-tasks.amazonaws.com"
}
}
},
{
"Action": "iam:PassRole",
"Effect": "Allow",
"Resource": ["arn:aws:iam::*:role/ecsInstanceRole*"
],
"Condition": {
"StringLike": {
"iam:PassedToService": ["ec2.amazonaws.com",
"ec2.amazonaws.com",
"ec2.amazonaws.com"
]}
}
}
AmazonEC2ContainerServiceFullAccess

This managed policy allows full administrator access to Amazon ECS.
AmazonEC2ContainerServiceforEC2Role

This managed policy allows Amazon ECS container instances to make calls to AWS on your behalf. For more information, see Amazon ECS Container Instance IAM Role (p. 415).

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:CreateCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:DiscoverPollEndpoint",
                "ecs:Poll",
                "ecs:RegisterContainerInstance",
                "ecs:StartTelemetrySession",
                "ecs:Submit*",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    }
}
```

AmazonEC2ContainerServiceRole

This managed policy allows Elastic Load Balancing load balancers to register and deregister Amazon ECS container instances on your behalf. For more information, see Amazon ECS Service Scheduler IAM Role (p. 426).

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:Describe*",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:Describe*",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets"
            ],
            "Resource": "*"
        }
    }
}
```
AmazonEC2ContainerServiceAutoscaleRole

This managed policy allows Application Auto Scaling to scale your Amazon ECS service's desired count up and down in response to CloudWatch alarms on your behalf. For more information, see Amazon ECS Service Auto Scaling IAM Role (p. 431).

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1456535218000",
            "Effect": "Allow",
            "Action": [
                "ecs: DescribeServices",
                "ecs: UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "Stmt1456535243000",
            "Effect": "Allow",
            "Action": [
                "cloudwatch: DescribeAlarms"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AmazonEC2ContainerServiceTaskRole

This IAM trust relationship policy allows containers in your Amazon ECS tasks to make calls to the AWS APIs on your behalf. For more information, see Amazon ECS Task Role (p. 432).

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts: AssumeRole"
        }
    ]
}
```

AmazonEC2ContainerServiceEventsRole

This policy allows CloudWatch Events to run tasks on your behalf. For more information, see Scheduled Tasks (cron) (p. 283).
Amazon ECR Managed Policies

Amazon ECR provides several managed policies that you can attach to IAM users or EC2 instances that allow differing levels of control over Amazon ECR 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 each API operation mentioned in these policies, see Actions in the Amazon Elastic Container Registry API Reference.

Topics

- AmazonEC2ContainerRegistryFullAccess (p. 414)
- AmazonEC2ContainerRegistryPowerUser (p. 414)
- AmazonEC2ContainerRegistryReadOnly (p. 415)

AmazonEC2ContainerRegistryFullAccess

This managed policy allows full administrator access to Amazon ECR.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["ecs:RunTask"],
      "Resource": ["*"]
    }
  ]
}
```

AmazonEC2ContainerRegistryPowerUser

This managed policy allows power user access to Amazon ECR, which allows read and write access to repositories, but does not allow users to delete repositories or change the policy documents applied to them.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["ecr:*"],
      "Resource": "*
    }
  ]
}
```
Amazon ECS Container Instance IAM Role

The Amazon ECS container agent makes calls to the Amazon ECS API on your behalf. Container instances that run the agent require an IAM policy and role for the service to know that the agent belongs to you. Before you can launch container instances and register them into a cluster, you must create an IAM role for those container instances to use when they are launched. This requirement applies to container instances launched with the Amazon ECS-optimized AMI provided by Amazon, or with any other instances that you intend to run the agent on.

**Important**

This role only applies if you are using the EC2 launch type.

**Important**

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 provided in the managed AmazonEC2ContainerServiceforEC2Role policy shown below.
If the containers in your tasks need extra permissions that are not listed here, we recommend providing those tasks with their own IAM roles. For more information, see IAM Roles for Tasks (p. 435).

You can prevent containers on the docker0 bridge from accessing the permissions supplied to the container instance role (while still allowing the permissions that are provided by IAM Roles for Tasks (p. 435)) by running the following `iptables` command on your container instances; however, containers will not be able to query instance metadata with this rule in effect. Note that this command assumes the default Docker bridge configuration and it will not work for containers that use the host network mode. For more information, see Network Mode (p. 196).

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 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
```

The AmazonEC2ContainerServiceforEC2Role policy is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:DiscoverPollEndpoint",
        "ecs:Poll",
        "ecs:RegisterContainerInstance",
        "ecs:StartTelemetrySession",
        "ecs:Submit*",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

**Note**

The `ecs:CreateCluster` line in the above policy is optional, provided that the cluster you intend to register your container instance into already exists. If the cluster does not already exist, the agent must have permission to create it, or you can create the cluster with the `create-cluster` command prior to launching your container instance.
If you omit the `ecs:CreateCluster` line, the Amazon ECS container agent can not create clusters, including the default cluster.

The `ecs:Poll` line in the above policy is used to grant the agent permission to connect with the Amazon ECS service to report status and get commands.

The Amazon ECS instance role is automatically created for you in the console first-run experience. However, you should manually 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 check and see if your account already has the Amazon ECS instance role and to attach the managed IAM policy if needed.

**To check for the `ecsInstanceRole` in the IAM console**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Search the list of roles for `ecsInstanceRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.
4. Choose the Permissions tab.
5. In the Managed Policies section, ensure that the `AmazonEC2ContainerServiceforEC2Role` managed policy is attached to the role. If the policy is attached, your Amazon ECS instance role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose Attach Policy.
   b. In the Filter box, type `AmazonEC2ContainerServiceforEC2Role` to narrow the available policies to attach.
   c. Check the box to the left of the `AmazonEC2ContainerServiceforEC2Role` policy and choose Attach Policy.
6. Choose the Trust Relationships tab, and Edit Trust Relationship.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose Cancel. If the trust relationship does not match, copy the policy into the Policy Document window and choose Update Trust Policy.

```json
{
    "Version": "2008-10-17",
    "Statement": [      
        {  
            "Sid": ",",
            "Effect": "Allow",
            "Principal": {  
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

**To create the `ecsInstanceRole` IAM role for your container instances**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles and then choose Create role.
3. Choose the AWS service role type, and then choose Elastic Container Service.
4. Choose the EC2 Role for Elastic Container Service use case and then Next: Permissions.
5. In the Attached permissions policy section, select `AmazonEC2ContainerServiceforEC2Role` and then choose Next: Review.
6. For **Role name**, type `ecsInstanceRole` and optionally you can enter a description.
7. Review your role information and then choose **Create role** to finish.

## Adding Amazon S3 Read-only Access to your Container Instance Role

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 Container Instance Configuration in Amazon S3 (p. 163).

### To allow Amazon S3 read-only access for your container instance role

2. In the navigation pane, choose **Roles**.
3. Choose the IAM role you use for your container instances (this role is likely titled `ecsInstanceRole`). For more information, see Amazon ECS Container Instance IAM Role (p. 415).
4. Choose the **Permissions** tab, then **Attach policy**.
5. On the **Attach policy** page, type `S3` into the **Filter: Policy type** field to narrow the policy results.
6. Check the box to the left of the **AmazonS3ReadOnlyAccess** policy and click **Attach policy**.

**Note**  
This policy allows read-only access to all Amazon S3 resources. For more restrictive bucket policy examples, see Bucket Policy Examples in the Amazon Simple Storage Service Developer Guide.

## Amazon ECS Task Execution IAM Role

The Amazon ECS container agent makes calls to the Amazon ECS API on your behalf, so it requires an IAM policy and role for the service to know that the agent belongs to you. The following are the use cases for when the task execution IAM role is needed:

- Your task uses the Fargate launch type and...
  - is pulling a container image from Amazon ECR.
  - uses the awsglue log driver.
- Your tasks uses either the Fargate or EC2 launch type and...
  - is using private registry authentication. For more information, see Required IAM Permissions for Private Registry Authentication (p. 420).
  - the task definition is referencing sensitive data using Secrets Manager secrets or AWS Systems Manager Parameter Store parameters. For more information, see Required IAM Permissions for Amazon ECS Secrets (p. 420).

**Note**  
The task execution role is supported by Amazon ECS container agent version 1.16.0 and later.

The `AmazonECSTaskExecutionRolePolicy` policy is shown below.
The Amazon ECS task execution role is automatically created for you in the Amazon ECS console first-run experience; however, you should 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 the following procedure to check and see if your account already has the Amazon ECS task execution role and to attach the managed IAM policy if needed.

**To check for the `ecsTaskExecutionRole` in the IAM console**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for **ecsTaskExecutionRole**. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**. Ensure that the **AmazonECSTaskExecutionRolePolicy** managed policy is attached to the role. If the policy is attached, your Amazon ECS task execution role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose **Attach policy**.
   b. To narrow the available policies to attach, for **Filter**, type **AmazonECSTaskExecutionRolePolicy**.
   c. Check the box to the left of the **AmazonECSTaskExecutionRolePolicy** policy and choose **Attach policy**.
5. Choose **Trust relationships**, **Edit trust relationship**.
6. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            }
        }
    ]
}
```
To create the `ecsTaskExecutionRole` IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create role.
3. In the Select type of trusted entity section, choose Elastic Container Service.
4. For Select your use case, choose Elastic Container Service Task, then choose Next: Permissions.
5. In the Attach permissions policy section, search for AmazonECSTaskExecutionRolePolicy, select the policy, and then choose Next: Review.
6. For Role Name, type `ecsTaskExecutionRole` and choose Create role.

Required IAM Permissions for Private Registry Authentication

The Amazon ECS task execution role is required to use the private registry authentication feature. This allows the container agent to pull the container image. For more information, see Private Registry Authentication for Tasks (p. 251).

To provide access to the secrets that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see Adding and Removing IAM Policies.

- `secretsmanager:GetSecretValue`
- `kms:Decrypt`—Required only if your key uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

An example inline policy adding the permissions is shown below.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:kms:<region>:<aws_account_id>:key:key_id"
      ]
    }
  ]
}
```

Required IAM Permissions for Amazon ECS Secrets

To use the Amazon ECS secrets feature, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary AWS Systems Manager or Secrets Manager resources. For more information, see Specifying Sensitive Data (p. 254).

To provide access to the AWS Systems Manager Parameter Store parameters that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see Adding and Removing IAM Policies.
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. Service-linked roles are predefined by Amazon ECS and include 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.

You can delete the roles only after first deleting their related resources. This protects your Amazon ECS resources because you can't inadvertently remove permission to access the resources.

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 Role 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 – Role to enable Amazon ECS to manage your cluster.

The AWSServiceRoleForECS service-linked role trusts the following services to assume the role:

- ssm:GetParameters—Required if you are referencing a Systems Manager Parameter Store parameter in a task definition.
- 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.
- kms:Decrypt—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions:

```json
{
  "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:kms:<region>:<aws_account_id>:key/key_id"
      ]
    }
  ]
}
```
The role permissions policy allows Amazon ECS to complete the following actions on the specified resources:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ECSTaskManagement",
      "Effect": "Allow",
      "Action": [
        "ec2:AttachNetworkInterface",
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:Describe*",
        "ec2:DetachNetworkInterface",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:DeregisterTargets",
        "elasticloadbalancing:Describe*",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:RegisterTargets",
        "route53:ChangeResourceRecordSets",
        "route53:CreateHealthCheck",
        "route53:DeleteHealthCheck",
        "route53:Get*",
        "route53:List*",
        "route53:UpdateHealthCheck",
        "servicediscovery:DeregisterInstance",
        "servicediscovery:Get*",
        "servicediscovery:List*",
        "servicediscovery:RegisterInstance",
        "servicediscovery:UpdateInstanceCustomHealthStatus"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ECSTagging",
      "Effect": "Allow",
      "Action": ["ec2:CreateTags"],
      "Resource": "arn:aws:ec2::*:*:network-interface/*"
    }
  ]
}
```

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.

**To allow an IAM entity to create the AWSServiceRoleForECS service-linked role**

Add the following statement to the permissions policy for the IAM entity that needs to create the service-linked role:

```json
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole",
    "iam:PutRolePolicy"
  ],
  "Condition": {
    "StringEquals": {
      "iam:PrincipalService": "elasticcontainerIService.amazonaws.com"
    }
  }
}
```
To allow an IAM entity to edit the description of the AWSServiceRoleForECS service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to edit the description of a service-linked role:

```json
{
    "Effect": "Allow",
    "Action": [
        "iam:UpdateRoleDescription"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ecs.amazonaws.com/AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}
```

To allow an IAM entity to delete the AWSServiceRoleForECS service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to delete a service-linked role:

```json
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ecs.amazonaws.com/AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}
```

Creating a Service-Linked Role for Amazon ECS

Under most circumstances, you don't need to manually create a service-linked role. For example, when you create a new cluster (for example, with the Amazon ECS first run, the cluster creation wizard, or the AWS CLI or SDKs), or create or update a service in the AWS Management Console, Amazon ECS creates the service-linked role for you, if it does not already exist.

**Important**
The IAM entity that is creating the cluster must have the appropriate IAM permissions to create the service-linked role and apply a policy to it. Otherwise, the automatic creation fails.

Creating a Service-Linked Role in IAM (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to create a service-linked role with the trust policy and inline policies that the service needs to assume the role.

**To create a service-linked role (CLI)**

Use the following command:

```
$ aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```
Editing a Service-Linked Role for Amazon ECS

Amazon ECS does not 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. You can, however, edit the description of the role. For more information, see Modifying a Role in the IAM User Guide.

Deleting a Service-Linked Role for Amazon ECS

If you no longer use Amazon ECS, we recommend that you delete the role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must delete all Amazon ECS clusters in all regions before you can delete the service-linked role.

Cleaning up a Service-Linked Role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and delete all Amazon ECS clusters in all AWS Regions.

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 a Service (p. 350) and Deleting a Service (p. 353).
2. Force deregister all container instances from all clusters in all regions. For more information, see Deregister a Container Instance (p. 131).
3. Delete all Amazon ECS clusters in all regions. For more information, see Deleting a Cluster (p. 59).

Deleting a Service-Linked Role in IAM (Console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (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. Then select the check box next to AWSServiceRoleForECS, not the name or row itself.
3. For **Role actions** at the top of the page, choose **Delete role**.

4. In the confirmation dialog box, review the service last accessed data, which shows when each of the
selected roles last accessed an AWS service. This helps you to confirm whether the role is currently
active. If you want to proceed, choose **Yes, Delete** to submit the service-linked role for deletion.

5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion.
Because the IAM service-linked role deletion is asynchronous, after you submit the role for deletion,
the deletion task can succeed or fail.

   - If the task succeeds, then the role is removed from the list and a notification of success appears at
     the top of the page.
   - If the task fails, you can choose **View details** or **View Resources** from the notifications to learn
     why the deletion failed. If the deletion fails because the role is using the service's resources, then
     the notification includes a list of resources, if the service returns that information. You can then
     clean up the resources and submit the deletion again.

   **Note**
   You might have to repeat this process several times, depending on the information that
   the service returns. For example, your service-linked role might use six resources and your
   service might return information about five of them. If you clean up the five resources
   and submit the role for deletion again, the deletion fails and the service reports the one
   remaining resource. A service might return all of the resources, a few of them, or it might
   not report any resources.

   - If the task fails and the notification does not include a list of resources, then the service might not
     return that information. To learn how to clean up the resources for that service, see **AWS Services
     That Work with IAM**. Find your service in the table, and choose the **Yes** link to view the service-
     linked role documentation for that service.

### Deleting a Service-Linked Role in IAM (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

**To delete a service-linked role (CLI)**

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you
must submit a deletion request. That request can be denied if these conditions are not met. You
must capture the **deletion-task-id** from the response to check the status of the deletion task.
Type the following command to submit a service-linked role deletion request:

   ```
   # aws iam delete-service-linked-role --role-name AWSServiceRoleForECS+OPTIONAL-SUFFIX
   ```

2. Type the following command to check the status of the deletion task:

   ```
   # aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
   ```

   The status of the deletion task can be **NOT_STARTED**, **IN_PROGRESS**, **SUCCEEDED**, or **FAILED**.
If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the
deletion fails because the role is using the service's resources, then the notification includes a list of
resources, if the service returns that information. You can then clean up the resources and submit
the deletion again.

   **Note**
   You might have to repeat this process several times, depending on the information that
   the service returns. For example, your service-linked role might use six resources and your
   service might return information about five of them. If you clean up the five resources
   and submit the role for deletion again, the deletion fails and the service reports the one
   remaining resource. A service might return all of the resources, a few of them, or it might
not report any resources. To learn how to clean up the resources for a service that does not report any resources, see AWS Services That Work with IAM. Find your service in the table, and choose the Yes link to view the service-linked role documentation for that service.

Deleting a Service-Linked Role in IAM (AWSAPI)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked roll, call DeleteServiceLinkedRole. In the request, specify the AWSServiceRoleForECS role name.

   Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the DeletionTaskId from the response to check the status of the deletion task.

2. To check the status of the deletion, call GetServiceLinkedRoleDeletionStatus. In the request, specify the DeletionTaskId.

   The status of the deletion task can be NOT_STARTED, IN_PROGRESS, SUCCEEDED, or FAILED. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot. If the deletion fails because the role is using the service's resources, then the notification includes a list of resources, if the service returns that information. You can then clean up the resources and submit the deletion again.

   **Note**
   
   You might have to repeat this process several times, depending on the information that the service returns. For example, your service-linked role might use six resources and your service might return information about five of them. If you clean up the five resources and submit the role for deletion again, the deletion fails and the service reports the one remaining resource. A service might return all of the resources, a few of them, or it might not report any resources. To learn how to clean up the resources for a service that does not report any resources, see AWS Services That Work with IAM. Find your service in the table, and choose the Yes link to view the service-linked role documentation for that service.

Amazon ECS Service Scheduler IAM Role

The Amazon ECS service scheduler makes calls to the Amazon EC2 and Elastic Load Balancing APIs on your behalf to register and deregister container instances with your load balancers. Before you can attach a load balancer to an Amazon ECS service, you must create an IAM role for your services to use before you start them. This requirement applies to any Amazon ECS service that you plan to use with a load balancer.

In most cases, the Amazon ECS service role is created for you automatically in the console first-run experience. You can use the following procedure to check if your account already has the Amazon ECS service role.

The AmazonEC2ContainerServiceRole policy is shown below.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AuthorizeSecurityGroupIngress"
      ]
    }
  ]
}
```
Note
The ec2:AuthorizeSecurityGroupIngress rule is reserved for future use. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To check for the ecsServiceRole in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Search the list of roles for ecsServiceRole. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.
4. Choose the Permissions tab.
5. In the Managed Policies section, ensure that the AmazonEC2ContainerServiceRole managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose Attach Policy.
   b. To narrow the available policies to attach, for Filter, type AmazonEC2ContainerServiceRole.
   c. Check the box to the left of the AmazonEC2ContainerServiceRole policy and choose Attach Policy.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose Cancel. If the trust relationship does not match, copy the policy into the Policy Document window and choose Update Trust Policy.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To create an IAM role for your service scheduler load balancers

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create role.
3. In the Select type of trusted entity section, choose Elastic Container Service.
4. In the **Select your use case** section, choose **Elastic Container Service** and choose **Next: Permissions**.
5. In the **Attached permissions policy** section, select the **AmazonEC2ContainerServiceRole** policy and choose **Next: Review**.
6. For **Role Name**, type `ecsServiceRole`, enter a **Role description** and then choose **Create role**.

### 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**

IAM users also require permissions to use CodeDeploy; these permissions are described in Blue/Green Deployment Required IAM Permissions (p. 301).

There are two managed policies provided. The **AWSCodeDeployRoleForECS** policy, shown below, gives CodeDeploy permission to update any resource using the associated action.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecs:DescribeServices",
        "ecs:CreateTaskSet",
        "ecs:UpdateServicePrimaryTaskSet",
        "ecs:DeleteTaskSet",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeListeners",
        "elasticloadbalancing:ModifyListener",
        "elasticloadbalancing:DescribeRules",
        "elasticloadbalancing:ModifyRule",
        "lambda:InvokeFunction",
        "cloudwatch:DescribeAlarms",
        "sns:Publish",
        "s3:GetObject",
        "s3:GetObjectMetadata",
        "s3:GetObjectVersion"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

The **AWSCodeDeployRoleForECSLimited** policy, shown below, gives CodeDeploy more limited permissions.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecs:DescribeServices",
        "ecs:CreateTaskSet",
        "ecs:UpdateServicePrimaryTaskSet",
        "ecs:DeleteTaskSet",
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```
To create an IAM role for CodeDeploy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles, Create role.
3. For Select type of trusted entity section, choose AWS service.
4. For Choose the service that will use this role, choose CodeDeploy.
5. For Select your use case, choose CodeDeploy, Next: Permissions.
6. Choose Next: Tags.
7. For Add tags (optional), you can add optional IAM tags to the role. Choose Next:Review when finished.
8. For Role name, type ecsCodeDeployRole, enter an optional description, and then choose Create role.

To add the required permissions to the Amazon ECS CodeDeploy IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Search the list of roles for *ecsCodeDeployRole*. If the role does not exist, use the procedure above to create the role. If the role does exist, select the role to view the attached policies.

3. In the Permissions policies section, ensure that either the *AWSCodeDeployRoleForECS* or *AWSCodeDeployRoleForECSLimited* managed policy is attached to the role. If the policy is attached, your Amazon ECS CodeDeploy service role is properly configured. If not, follow the substeps below to attach the policy.

   a. Choose Attach policies.
   b. To narrow the available policies to attach, for Filter, type *AWSCodeDeployRoleForECS* or *AWSCodeDeployRoleForECSLimited*.
   c. Check the box to the left of the AWS managed policy and choose Attach policy.


5. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose Cancel. If the trust relationship does not match, copy the policy into the Policy Document window and choose Update Trust Policy.

   { 
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": ",",
           "Effect": "Allow",
           "Principal": {
               "Service": ["codedeploy.amazonaws.com"]
           },
           "Action": "sts:AssumeRole"
       }
   ]
   }

6. If the tasks in your Amazon ECS service using the blue/green deployment type require the use of the task execution role or a task role override, then you must add the *iam:PassRole* permission for each task execution role or task role override to the CodeDeploy IAM role as an inline policy. For more information, see Amazon ECS Task Execution IAM Role (p. 418) and Amazon ECS Task Role (p. 432).

   Follow the substeps below to create an inline policy.

   b. Search the list of roles for *ecsCodeDeployRole*. If the role does not exist, use the procedure above to create the role. If the role does exist, select the role to view the attached policies.
   c. In the Permissions policies section, choose Add inline policy.
   d. Choose the JSON tab and add the following policy text.

   `{ 
   "Version": "2012-10-17",
   "Statement": [ 
       { 
           "Effect": "Allow",
           "Action": "iam:PassRole",
           "Resource": ["arn:aws:iam::<aws_account_id>:role/<ecsTaskExecutionRole_or_TaskRole_name>"
       ]
       
   ]
   `}
Note
Specify the full ARN of your task execution role or task role override.

e. Choose Review policy
f. For Name, type a name for the added policy and then choose Create policy.

Amazon ECS Service Auto Scaling IAM Role

Before you can use Service Auto Scaling with Amazon ECS, the Application Auto Scaling service needs permissions to describe your CloudWatch alarms and registered services, as well as permissions to update your Amazon ECS service's desired count on your behalf. These permissions are provided by the Service Auto Scaling IAM role (ecsAutoscaleRole).

Note
IAM users also require permissions to use Service Auto Scaling; these permissions are described in Service Auto Scaling Required IAM Permissions (p. 326). If an IAM user has the required permissions to use Service Auto Scaling in the Amazon ECS console, create IAM roles, and attach IAM role policies to them, then that user can create this role automatically as part of the Amazon ECS console create service (p. ______) or update service (p. 350) workflows, and then use the role for any other service later (in the console or with the AWS CLI or SDKs).

You can use the following procedure to check and see if your account already has Service Auto Scaling.

The AmazonEC2ContainerServiceAutoscaleRole policy is shown below.

```json
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "Stmt1456535218000",
"Effect": "Allow",
"Action": [
"ecs:DescribeServices",
"ecs:UpdateService"
],
"Resource": ["
"
]
},
{
"Sid": "Stmt1456535243000",
"Effect": "Allow",
"Action": [
"cloudwatch:DescribeAlarms"
],
"Resource": ["
"
]
}
]
}
```

To check for the Service Auto Scaling role in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Search the list of roles for `ecsAutoscaleRole`. If the role does not exist, use the procedure below to create the role. If the role does exist, select the role to view the attached policies.

4. Choose the Permissions tab.

5. In the Managed Policies section, ensure that the `AmazonEC2ContainerServiceAutoscaleRole` managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose Attach Policy.
   b. To narrow the available policies to attach, for Filter, type `AmazonEC2ContainerServiceAutoscaleRole`.
   c. Select the box to the left of the `AmazonEC2ContainerAutoscaleRole` policy and choose Attach Policy.


7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose Cancel. If the trust relationship does not match, copy the policy into the Policy Document window and choose Update Trust Policy.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "application-autoscaling.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

To create an IAM role for Service Auto Scaling

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles and then choose Create New Role.
3. In the Select Role Type section, scroll down and choose Select next to the Amazon Elastic Container Service Autoscale Role service role.
4. In the Attach Policy section, select the `AmazonEC2ContainerServiceAutoscaleRole` policy and then choose Next Step.
5. In the Role Name field, type `ecsAutoscaleRole` to name the role, and then choose Next Step.
6. Review your role information and then choose Create Role to finish.

Amazon ECS Task Role

Before you can use IAM roles for tasks, Amazon ECS needs permission to make calls to the AWS APIs on your behalf. These permissions are provided by the Amazon ECS Task Role.

You can create a task IAM role for each task definition that needs permission to call AWS APIs. You simply create an IAM policy that defines which permissions your task should have, and then attach that policy to a role that uses the Amazon ECS Task Role trust relationship policy. For more information, see Creating an IAM Role and Policy for your Tasks (p. 437).

The Amazon ECS Task Role trust relationship is shown below.
CloudWatch Events IAM Role

Before you can use Amazon ECS scheduled tasks with CloudWatch Events rules and targets, the CloudWatch Events service needs permissions to run Amazon ECS tasks on your behalf. These permissions are provided by the CloudWatch Events IAM role (ecsEventsRole).

The CloudWatch Events role is automatically created for you in the AWS Management Console when you configure a scheduled task. For more information, see Scheduled Tasks (cron) (p. 283).

The AmazonEC2ContainerServiceEventsRole policy is shown below.

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": ["ecs:RunTask"],
"Resource": ["*"],
"Principal": {
"Service": "ecs-tasks.amazonaws.com"
}
}
]
}
```

If your scheduled tasks require the use of the task execution role or a task role override, then you must add `iam:PassRole` permissions for each task execution role or task role override to the CloudWatch IAM role. For more information about the task execution role, see Amazon ECS Task Execution IAM Role (p. 418).

**Note**
Specify the full ARN of your task execution role or task role override.

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "iam:PassRole",
"Resource": ["arn:aws:iam::<aws_account_id>:role/<ecsTaskExecutionRole_or_TaskRole_name>"
}
]
}
```
You can use the following procedure to check that your account already has the CloudWatch Events IAM role, and manually create it if needed.

**To check for the CloudWatch Events IAM role in the IAM console**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsEventsRole`. If the role does not exist, use the next procedure to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**.
5. In the **Managed Policies** section, ensure that the `AmazonEC2ContainerServiceEventsRole` managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.
   a. Choose **Attach Policy**.
   b. To narrow the available policies to attach, for **Filter**, type `AmazonEC2ContainerServiceEventsRole`.
   c. Select the box to the left of the `AmazonEC2ContainerServiceEventsRole` policy and choose **Attach Policy**.
6. Choose **Trust Relationships, Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

**To create an IAM role for CloudWatch Events**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. In the **Select type of trusted entity** section, choose **Elastic Container Service**. For **Select your use case** choose **Elastic Container Service Task**. Choose **Next: Permissions**.
4. In the **Attach permissions policy** section, select the `AmazonEC2ContainerServiceEventsRole` policy and choose **Next: Review**.
5. For **Role name**, type `ecsEventsRole` to name the role, optionally enter a description, and then choose **Create role**.
6. Review your role information and choose **Create Role**.
IAM Roles for Tasks

With IAM roles for Amazon ECS tasks, you can specify an IAM role that can be used by the containers in a task. Applications must sign their AWS API requests with AWS credentials, and this feature provides a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the EC2 instance’s role, you can associate an IAM role with an ECS task definition or RunTask API operation. The applications in the task’s containers can then use the AWS SDK or CLI to make API requests to authorized AWS services.

Important
Containers that are running on your container instances are not prevented from accessing the credentials that are supplied to the container instance profile (through the Amazon EC2 instance metadata server). We recommend that you limit the permissions in your container instance role to the minimal list of permissions shown in Amazon ECS Container Instance IAM Role (p. 415).

To prevent containers in tasks that use the awsvpc network mode from accessing the credential information supplied to the container 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 (p. 152).

To prevent containers in tasks that use the bridge network mode from accessing the credential information supplied to the container instance profile (while still allowing the permissions that are provided by the task role) by running the following iptables command on your container instances. Note that this command does not affect containers in tasks that use the host or awsvpc network modes. For more information, see Network Mode (p. 196).

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 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:

  ```bash
  sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
  ```

- For the Amazon ECS-optimized Amazon Linux AMI:

  ```bash
  sudo service iptables save
  ```

You define the IAM role to use in your task definitions, or you can use a `taskRoleArn` override when running a task manually with the `RunTask` API operation. The Amazon ECS agent receives a payload message for starting the task with additional fields that contain the role credentials. The Amazon ECS agent sets a unique task credential ID as an identification token and updates its internal credential cache so that the identification token for the task points to the role credentials that are received in the payload. The Amazon ECS agent populates the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable in the `Env` object (available with the `docker inspect container_id` command) for all containers that belong to this task with the following relative URI: `/credential_provider_version/credentials?id=task_credential_id`.

**Note**

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 no longer inherit any IAM permissions from the container instance.

From inside the container, you can query the credentials with the following command:

```bash
curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

**Output:**

```json
{
    "AccessKeyId": "ACCESS_KEY_ID",
    "Expiration": "EXPIRATION_DATE",
    "RoleArn": "TASK_ROLE_ARN",
    "SecretAccessKey": "SECRET_ACCESS_KEY",
    "Token": "SECURITY_TOKEN_STRING"
}
```

If your container instance is using at least version 1.11.0 of the container agent and a supported version of the AWS CLI or SDKs, then the SDK client will see that the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` variable is available, and it will use the provided credentials to make calls to the AWS APIs. For more information, see [Enabling Task IAM Roles on your Container Instances](p. 437) and [Using a Supported AWS SDK](p. 439).

Each time the credential provider is used, the request is logged locally on the host container instance at `/var/log/ecs/audit.log.YYYY-MM-DD-HH`. For more information, see [IAM Roles for Tasks Credential Audit Log](p. 601).

**Topics**

- Benefits of Using IAM Roles for Tasks (p. 437)
- Enabling Task IAM Roles on your Container Instances (p. 437)
- Creating an IAM Role and Policy for your Tasks (p. 437)
Benefits of Using IAM Roles for Tasks

- **Credential Isolation**: A container can only retrieve credentials for the IAM role that is defined in the task definition to which it belongs; a container never has access to credentials that are intended for another container that belongs to another task.
- **Authorization**: Unauthorized containers cannot access IAM role credentials defined for other tasks.
- **Auditability**: Access and event logging is 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 is using which role.

Enabling Task IAM Roles on your Container Instances

Your Amazon ECS container instances require at least version 1.11.0 of the container agent to enable 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 (p. 146). If you are using the Amazon ECS-optimized AMI, your instance needs at least 1.11.0-1 of the `ecs-init` package. If your container instances are launched from version 2016.03.e or later, then they contain the required versions of the container agent and `ecs-init`. For more information, see Amazon ECS-optimized AMIs (p. 62).

If you are not using the Amazon ECS-optimized AMI for your container instances, be sure to add the `--net=host` option to your `docker run` command that starts the agent and the appropriate agent configuration variables for your desired configuration (for more information, see Amazon ECS Container Agent Configuration (p. 152)):

```
ECS_ENABLE_TASK_IAM_ROLE=true
```

Enables IAM roles for tasks for containers with the `bridge` and `default` network modes.

```
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
```

Enables 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) (p. 150). 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 --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.

Creating an IAM Role and Policy for your Tasks

You must create an IAM policy for your tasks to use that specifies the permissions that you would like the containers in your tasks to have. You have several ways to create a new IAM permission policy. You
can copy a complete AWS managed policy that already does some of what you’re looking for and then customize it to your specific requirements. For more information, see Creating a New Policy in the IAM User Guide.

You must also create a role for your tasks to use before you can specify it in your task definitions. You can create the role using the Amazon Elastic Container Service Task Role service role 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.

**Note**
To view the trust relationship for this role, see Amazon ECS Task Role (p. 432).

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.

**To create an IAM policy for your tasks**

In this example, we create a policy to allow read-only access to an Amazon S3 bucket. You could store database credentials or other secrets in this bucket, and the containers in your task can read the credentials from the bucket and load them into your application.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Policies and then choose Create policy.
3. Follow the steps under one of the following tabs, which shows you how to use the visual or JSON editors.

Using the visual editor

1. For Service, choose S3.
2. For Actions, expand the Read option and select GetObject.
3. For Resources, select Add ARN and enter the full ARN of your Amazon S3 bucket.
5. In the Review policy section, for Name type your own unique name, such as AmazonECSTaskS3BucketPolicy.
6. Choose Create policy to finish.

Using the JSON editor

1. In the Policy Document field, paste the policy to apply to your tasks. The example below allows permission to the my-task-secrets-bucket Amazon S3 bucket. You can modify the policy document to suit your specific needs.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-task-secrets-bucket/*
            ]
        }
    ]
}
```

API Version 2014-11-13
438
2. Choose **Create Policy**.

**To create an IAM role for your tasks**

2. In the navigation pane, choose **Roles, Create New Role**.
3. In the **Select Role Type** section, for the **Amazon Elastic Container Service Task Role** service role, choose **Select**.
   
   **Note**
   
   To view the trust relationship for this role, see **Amazon ECS Task Role** (p. 432).
4. In the **Attach Policy** section, select the policy to use for your tasks (in this example **AmazonECSTaskS3BucketPolicy**), and then choose **Next Step**.
5. For **Role Name**, enter a name for your role. For this example, type **AmazonECSTaskS3BucketRole** to name the role, and then choose **Create Role** to finish.

**Using a Supported AWS SDK**

Support for IAM roles for tasks was added to the AWS SDKs on July 13th, 2016. The containers in your tasks must use an AWS SDK version that was created on or after that date. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature.

To ensure that you are using a supported SDK, follow the installation instructions for your preferred SDK at [Tools for Amazon Web Services](https://aws.amazon.com/tools/) when you are building your containers to get the latest version.

**Specifying an IAM Role for your Tasks**

After you have created a role and attached a policy to that role, you can run tasks that assume the role. You have several options to do this:

- Specify an IAM role for your tasks in the task definition. You can create a new task definition or a new revision of an existing task definition and specify the role you created previously. If you use the console to create your task definition, choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the **taskRoleArn** parameter. For more information, see **Creating a Task Definition** (p. 188).

  **Note**
  
  This option is required if you want to use IAM task roles in an Amazon ECS service.

- Specify an IAM task role override when running a task. You can specify an IAM task role override when running a task. If you use the console to run your task, choose **Advanced Options** and then choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the **taskRoleArn** parameter in the **overrides** JSON object. For more information, see **Running Tasks** (p. 269).

  **Note**
  
  In addition to the standard Amazon ECS permissions required to run tasks and services, IAM users also require **iam:PassRole** permissions to use IAM roles for tasks.

**Amazon ECS IAM Policy Examples**

The following examples show policy statements that you could use to control the permissions that IAM users have for Amazon ECS.
Amazon ECS First Run Wizard

The Amazon ECS first-run wizard simplifies the process of creating a cluster and running your tasks and services. However, users require permissions to many API operations from multiple AWS services to complete the wizard. The AmazonECS_FullAccess (p. 408) managed policy below shows the required permissions to complete the Amazon ECS first-run wizard.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget",
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
                "application-autoscaling:PutScalingPolicy",
                "application-autoscaling:RegisterScalableTarget",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:DeleteLaunchConfiguration",
                "autoscaling:DescribeAutoScalingGroup",
                "autoscaling:DescribeLaunchConfiguration",
                "autoscaling:Describe*",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:PutMetricAlarm",
                "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",
            ]
        }
    ]
}
```

API Version 2014-11-13
440
"codedeploy:GetApplicationRevision",
"codedeploy:RegisterApplicationRevision",
"codedeploy:BatchGetApplicationRevisions",
"codedeploy:BatchGetDeploymentGroups",
"codedeploy:BatchGetDeployments",
"codedeploy:BatchGetApplications",
"codedeploy:ListApplicationRevisions",
"codedeploy:ListDeploymentConfigs",
"codedeploy:ContinueDeployment",
"sns:ListTopics",
"lambda:ListFunctions",
"ec2:AssociateRouteTable",
"ec2:AttachInternetGateway",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:CancelSpotFleetRequests",
"ec2:CreateInternetGateway",
"ec2:CreateLaunchTemplate",
"ec2:CreateRoute",
"ec2:CreateRouteTable",
"ec2:CreateSecurityGroup",
"ec2:CreateSubnet",
"ec2:CreateVpc",
"ec2:DeleteLaunchTemplate",
"ec2:DeleteSubnet",
"ec2:DeleteVpc",
"ec2:Describe*",
"ec2:DetachInternetGateway",
"ec2:DisassociateRouteTable",
"ec2:ModifySubnetAttribute",
"ec2:ModifyVpcAttribute",
"ec2:RunInstances",
"ec2:RequestSpotFleet",
"elasticloadbalancing:CreateListener",
"elasticloadbalancing:CreateLoadBalancer",
"elasticloadbalancing:CreateRule",
"elasticloadbalancing:CreateTargetGroup",
"elasticloadbalancing:DeleteListener",
"elasticloadbalancing:DeleteLoadBalancer",
"elasticloadbalancing:DeleteRule",
"elasticloadbalancing:DeleteTargetGroup",
"elasticloadbalancing:DescribeListeners",
"elasticloadbalancing:DescribeLoadBalancers",
"elasticloadbalancing:DescribeRules",
"elasticloadbalancing:DescribeTargetGroups",
"ecs:*",
"events:DescribeRule",
"events:DeleteRule",
"events:ListRuleNamesByTarget",
"events:ListTargetsByRule",
"events:PutRule",
"events:PutTargets",
"events:RemoveTargets",
"iam:ListAttachedRolePolicies",
"iam:ListInstanceProfiles",
"iam:ListRoles",
"logs:CreateLogGroup",
"logs:DescribeLogGroups",
"logs:FilterLogEvents",
"route53:GetHostedZone",
"route53:ListHostedZonesByName",
"route53:CreateHostedZone",
"route53:DeleteHostedZone",
"route53:GetHealthCheck",
"servicediscovery:CreatePrivateDnsNamespace",
"servicediscovery:CreateService",
"servicediscovery:GetNamespace",  
0

API Version 2014-11-13
441
"servicediscovery:GetOperation",
"servicediscovery:GetService",
"servicediscovery:ListNamespaces",
"servicediscovery:ListServices",
"servicediscovery:UpdateService"
},
"Resource": [
  "*"
],
"Effect": "Allow",
"Action": [
  "ssm:GetParametersByPath",
  "ssm:GetParameters",
  "ssm:GetParameter"
],
"Resource": "arn:aws:ssm:*:*:parameter/aws/service/ecs*"
},
{
"Effect": "Allow",
"Action": [
  "ec2:DeleteInternetGateway",
  "ec2:DeleteRoute",
  "ec2:DeleteRouteTable",
  "ec2:DeleteSecurityGroup"
],
"Resource": [
  "*"
],
"Condition": {
  "StringLike": {
    "ec2:ResourceTag/aws:cloudformation:stack-name": "EC2ContainerService-"
  }
}
},
{
"Action": "iam:PassRole",
"Effect": "Allow",
"Resource": [
  "*"
],
"Condition": {
  "StringLike": {
    "iam:PassedToService": "ecs-tasks.amazonaws.com"
  }
}
},
{
"Action": "iam:PassRole",
"Effect": "Allow",
"Resource": [
  "arn:aws:iam::*:role/ecsInstanceRole*"
],
"Condition": {
  "StringLike": {
    "iam:PassedToService": [
      "ec2.amazonaws.com",
      "ec2.amazonaws.com.cn"
    ]
  }
}
}
The first run wizard also attempts to automatically create different IAM roles depending on the launch type of the tasks used. Examples are the Amazon ECS service role, container instance IAM role, and the task execution IAM role. To ensure that the first-run experience is able to create these IAM roles, one of the following must be true:

- Your user has administrator access. For more information, see Setting Up with Amazon ECS (p. 7).
- 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.
- You have a user with administrator access manually create the required IAM role so it is available on the account to be used. For more information, see the following:
  - Amazon ECS Service Scheduler IAM Role (p. 426)
  - Amazon ECS Container Instance IAM Role (p. 415)
  - Amazon ECS Task Execution IAM Role (p. 418)

## Clusters

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.
The following IAM policy allows permission to describe and delete a specific cluster. The DescribeCluster and DeleteCluster actions accept cluster ARNs as resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeCluster",
        "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": "*"
    }
  ]
}
```
Container Instances

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:

```json
{
  "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 *.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeContainerInstances"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
        }
      },
      ]
    }
  ]
}
```
Task Definitions

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:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "ecs:RegisterTaskDefinition",
            "ecs:ListTaskDefinitions",
            "ecs:DescribeTaskDefinition"
         ],
         "Resource": [
            "*
         ]
      }
   ]
}
```

Run Tasks

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 Tasks

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:container-instances": [
                        "arn:aws:ecs:<region>:<aws_account_id>:container-instance/<container_instance_UUID>"
                    ]
                },
                "Resource": ["arn:aws:ecs:<region>:<aws_account_id>:task-definition/<task_family>:*"
            ]
        }
    ]
}
```

**List and Describe Tasks**

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:DescribeTask"]
        }
    ]
}
```
Create Services

The following IAM policy allows a user to create Amazon ECS services in the AWS Management Console:

```json
{
    "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",
                "elasticsearch:Describe*",
                "iam:AttachRolePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:ListRoles",
                "iam:ListGroups",
                "iam:ListUsers"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Update Services

The following IAM policy allows a user to update Amazon ECS services in the AWS Management Console:

```json
{
    "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/<task_UUID>"
            ]
        }
    ]
}
```
Using the Amazon ECS Command Line Interface

The Amazon Elastic Container Service (Amazon ECS) command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment. The Amazon ECS CLI supports Docker Compose files (Version 1, Version 2, and Version 3), a popular open-source specification for defining and running multi-container applications. Use the CLI as part of your everyday development and testing cycle as an alternative to the AWS Management Console.

The latest version of the Amazon ECS CLI is 1.14.0. For release notes, see Changelog.

Note
The source code for the Amazon ECS CLI is available on GitHub. We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services does not currently support running modified copies of this software.

Topics
- Installing the Amazon ECS CLI (p. 450)
- Configuring the Amazon ECS CLI (p. 455)
- Migrating Configuration Files (p. 457)
- Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI (p. 458)
- Tutorial: Creating a Cluster with an EC2 Task Using the Amazon ECS CLI (p. 462)
- Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 468)
- Amazon ECS Command Line Reference (p. 471)

Installing the Amazon ECS CLI

Follow these instructions to install the Amazon ECS CLI on your macOS, Linux, or Windows system.

Step 1: Download the Amazon ECS CLI

Download the Amazon ECS CLI binary.

- For macOS:
  ```bash
  ```

- For Linux systems:
  ```bash
  ```

- For Windows systems:
  ```bash
  Open Windows PowerShell and run the following commands:

  PS C:\> New-Item ‘C:\Program Files\Amazon\ECSCLI’ -type directory
  ```
Step 2: (Optional) Verify the Amazon ECS CLI

To verify the validity of the Amazon ECS CLI file, you can either use the provided MD5 sum or the PGP signatures. Both methods are described in the following sections.

Verify Using the MD5 Sum

Verify the downloaded binary with the MD5 sum provided.

- For macOS (compare the two output strings to verify that they match):

  ```
  curl -s https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-amd64-latest.md5 && md5 -q /usr/local/bin/ecs-cli
  ```

- For Linux systems (look for an OK in the output string):

  ```
  echo "$(curl -s https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-linux-amd64-latest.md5) /usr/local/bin/ecs-cli" | md5sum -c -
  ```

- For Windows systems:

  Open Windows PowerShell and find the md5 hash of the executable that you downloaded:

  ```
  PS C:\> Get-FileHash ecs-cli.exe -Algorithm MD5
  ```

  Compare that with this md5 hash:

  ```
  PS C:\> Get-Content md5.txt
  ```

Verify Using the PGP Signature

The Amazon ECS CLI executables are cryptographically signed using PGP signatures. You can use the following steps to verify the signatures using the GnuPG tool.

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

   - For 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
     ```

   - For Linux systems, install gpg using the package manager on your flavor of Linux.
2. Retrieve the Amazon ECS PGP public key. You can use a command to do this or manually create the key and then import it.
   a. Option 1: Retrieve the key with the following command.
   ```bash
   gpg --keyserver hkp://keys.gnupg.net --recv BCE9D9A42D51784F
   ```
   b. Option 2: Create a file with the following contents of the Amazon ECS PGP public key and then import it:
   ```plaintext
   -----BEGIN PGP PUBLIC KEY BLOCK-----
   Version: GnuPG v2
   mQINBFq1SasBeAD1icrT1iVv1ydfn8DgqbYe9ne3t6jgKFmKowLm6LGLj7eH7j
   qJt0ghC0WDRmk+quPq9dGrDAz2npxY5fIkJhgar4CP8QgRnRMG2f1741mavrY4g
   7k/K8HVHlg2uW3/294XLqEgbR0TMdWFdxoPcTeBqAMj3Lgn6P6+e6vXVrhChQz
   BqOAhj immigrants 01jNqjNLUMAG56198LgPsvAmpuqC7kyZvP4ywWwARQABB
   2CRWB6F624gRUNITDxIK3Mc2vjdXpdlhjAV1hmem9uLmnVb76jAhWEACAAAY
   AIrljLOyAC09q1XvEs0Aqrgq/j+JppqPwHJnV7wvlesB1IS0ug26D6uPvHd7
   452
   ```

b. Option 2: Create a file with the following context of the Amazon ECS PGP public key and then import it:

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

mQINBFq1SasBeAD1icrT1iVv1ydfn8DgqbYe9ne3t6jgKFmKowLm6LGLj7eH7j
qJt0ghC0WDRmk+quPq9dGrDAz2npxY5fIkJhgar4CP8QgRnRMG2f1741mavrY4g
7k/K8HVHlg2uW3/294XLqEgbR0TMdWFdxoPcTeBqAMj3Lgn6P6+e6vXVrhChQz
BqOAhj immigrants 01jNqjNLUMAG56198LgPsvAmpuqC7kyZvP4ywWwARQABB
2CRWB6F624gRUNITDxIK3Mc2vjdXpdlhjAV1hmem9uLmnVb76jAhWEACAAAY
AIrljLOyAC09q1XvEs0Aqrgq/j+JppqPwHJnV7wvlesB1IS0ug26D6uPvHd7
452

```

```
API Version 4.13.2014-05-02
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

Import the Amazon ECS PGP public key with the following command.

```
gpg --import <public_key_filename>
```

3. Download the Amazon ECS 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.

   • For macOS systems:

     ```
```

   • For Linux systems:

     ```
```

   • For Windows systems:

     ```
```

4. Verify the signature.

   • For macOS and Linux systems:

     ```
gpg --verify ecs-cli.asc /usr/local/bin/ecs-cli
```

   • For Windows systems:

     ```
PS C:\> gpg --verify ecs-cli.asc 'C:\Program Files\Amazon\ECSCLI\ecs-cli.exe'
```

Expected output:

```
API Version 2014-11-13
454
```
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.

Step 3: Apply Execute Permissions to the Binary

Apply execute permissions to the binary.

- For macOS and Linux systems:

  ```bash
  sudo chmod +x /usr/local/bin/ecs-cli
  ```

- For Windows systems:

  Edit the environment variables and add `C:\Program Files\Amazon\ECSCLI` to the `PATH` variable field, separated from existing entries by using a semicolon. For example:

  ```powershell
  PS C:\> C:\existing\path;C:\Program Files\Amazon\ECSCLI
  ```

  Restart PowerShell (or the command prompt) so the changes go into effect.

  **Note**
  Once the `PATH` variable is set, the Amazon ECS CLI can be used from either Windows PowerShell or the command prompt.

Step 4: Complete the Installation

Verify that the CLI is working properly.

```
ecs-cli --version
```

Proceed to Configuring the Amazon ECS CLI (p. 455).

**Important**
You must configure the Amazon ECS CLI with your AWS credentials, an AWS region, and an Amazon ECS cluster name before you can use it.

Configuring the Amazon ECS CLI

The Amazon ECS CLI requires some basic configuration information before you can use it, such as your AWS credentials, the AWS Region in which to create your cluster, and the name of the Amazon ECS cluster to use. Configuration information is stored in the `~/.ecs` directory on macOS and Linux systems and in `C:\Users\<username>\AppData\local\ecs` on Windows systems.
To configure the Amazon ECS CLI

1. Set up a CLI profile with the following command, substituting profile_name with your desired profile name, $AWS_ACCESS_KEY_ID and $AWS_SECRET_ACCESS_KEY environment variables with your AWS credentials.

   ```bash
   ecs-cli configure profile --profile-name profile_name --access-key $AWS_ACCESS_KEY_ID --secret-key $AWS_SECRET_ACCESS_KEY
   ```

2. Complete the configuration with the following command, substituting launch_type with the task launch type you want to use by default, region_name with your desired AWS region, cluster_name with the name of an existing Amazon ECS cluster or a new cluster to use, and configuration_name for the name you’d like to give this configuration.

   ```bash
   ecs-cli configure --cluster cluster_name --default-launch-type launch_type --region region_name --config-name configuration_name
   ```

After you have installed and configured the CLI, you can try the Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI (p. 458). For more information, see the Amazon ECS Command Line Reference in the Amazon Elastic Container Service Developer Guide.

Profiles

The Amazon ECS CLI supports the configuring of multiple sets of AWS credentials as named profiles using the `ecs-cli configure profile` command. A default profile can be set by using the `ecs-cli configure profile default` command. These profiles can then be referenced when you run Amazon ECS CLI commands that require credentials using the `--ecs-profile` flag otherwise the default profile is used.

For more information, see the Amazon ECS Command Line Reference in the Amazon Elastic Container Service Developer Guide.

Cluster Configurations

A cluster configuration is a set of fields that describes an Amazon ECS cluster including the name of the cluster and the region. A default cluster configuration can be set by using the `ecs-cli configure default` command. The Amazon ECS CLI supports the configuring of multiple named cluster configurations using the `--config-name` option.

For more information, see the Amazon ECS Command Line Reference in the Amazon Elastic Container Service Developer Guide.

Order of Precedence

There are multiple methods for passing both the credentials and the region in an Amazon ECS CLI command. The following is the order of precedence for each of these.

The order of precedence for credentials is:

1. Amazon ECS CLI profile flags:
   a. ECS profile (`--ecs-profile`)
   b. AWS profile (`--aws-profile`)
2. Environment variables:
   a. ECS_PROFILE
   b. AWS_PROFILE
Migrating Configuration Files

The process of configuring the Amazon ECS CLI has changed significantly in the latest version (v1.0.0) to allow the addition of new features. A migration command has been introduced that converts an older (v0.6.6 and older) configuration file to the current format. The old configuration files are deprecated, so we recommend converting your configuration to the newest format to take advantage of the new features. The configuration-related changes and new features introduced in v1.0.0 in the new YAML formatted configuration files include:

- Splitting up of credential and cluster-related configuration information into two separate files. Credential information is stored in `~/.ecs/credentials` and cluster configuration information is stored in `~/.ecs/config`.
- The configuration files are formatted in YAML.
- Support for storing multiple named configurations.
- Deprecation of the field `compose-service-name-prefix` (name used for creating a service `<compose_service_name_prefix> + <project_name>`). This field can still be configured. However, if it is not configured, there is no longer a default value assigned. For Amazon ECS CLI v0.6.6 and earlier, the default was `ecscompose-service-`.
- Removal of the field `compose-project-name-prefix` (name used for creating a task definition `<compose_project_name_prefix> + <project_name>`). Amazon ECS CLI v1.0.0 and later can still read old configuration files; so if this field is present then it is still read and used. However, configuring this field is not supported in v1.0.0+ with the `ecs-cli configure` command, and if the field is manually added to a v1.0.0+ configuration file it causes the Amazon ECS CLI to throw an error.
- The field `cfn-stack-name-prefix` (name used for creating CFN stacks `<cfn_stack_name_prefix> + <cluster_name>`) has been changed to `cfn-stack-name`. Instead of specifying a prefix, the exact name of a CloudFormation template can be configured.
- Amazon ECS CLI v0.6.6 and earlier allowed configuring credentials using a named AWS profile from the `~/.aws/credentials` file on your system. This functionality has been removed. However, a new
flag, --aws-profile, has been added which allows the referencing of an AWS profile inline in all commands that require credentials.

**Note**
The --project-name flag can be used to set the project name.

### Migrating Older Configuration Files to the v1.0.0+ Format

While all versions of the Amazon ECS CLI support reading from the older configuration file format, upgrading to the new format is required to take advantage of some new features, for example using multiple named cluster profiles. Migrating your legacy configuration file to the new format is easy with the `ecs-cli configure migrate` command. The command takes the configuration information stored in the old format in `~/.ecs/config` and converts it to a pair of files in the new format, overwriting your old configuration file in the process.

When running the `ecs-cli configure migrate` command there is a warning message displayed with the old configuration file, and a preview of the new configuration files. User confirmation is required before the migration proceeds. If the --force flag is used, then the warning message is not displayed, and the migration proceeds without any confirmation. If cfn-stack-name-prefix is used in the legacy file, then cfn-stack-name is stored in the new file as `<cfn_stack_name_prefix> + <cluster_name>`.

For more information, see the Amazon ECS Command Line Reference in the Amazon Elastic Container Service Developer Guide.

### Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI

This tutorial shows you how to set up a cluster and deploy a task using the Fargate launch type.

**Prerequisites**

Complete the following prerequisites:

- Set up an AWS account.
- Install the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 450).
- Install and configure the AWS CLI. For more information, see AWS Command Line Interface.

**Step 1: Create the Task Execution IAM Role**

Amazon ECS needs permissions so that your Fargate task can store logs in CloudWatch. These permissions are covered by the task execution IAM role. For more information, see Amazon ECS Task Execution IAM Role (p. 418).

**To create the task execution IAM role using the AWS CLI**

1. Create a file named `task-execution-assume-role.json` with the following contents:

   ```json
   {
   ```

   API Version 2014-11-13
   458
Step 2: Configure the Amazon ECS CLI

The Amazon ECS CLI requires credentials in order to make API requests on your behalf. It can pull credentials from environment variables, an AWS profile, or an Amazon ECS profile. For more information, see Configuring the Amazon ECS CLI (p. 455).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration, which defines the AWS region to use, resource creation prefixes, and the cluster name to use with the Amazon ECS CLI:

   ```bash
   ecs-cli configure --cluster tutorial --region us-east-1 --default-launch-type FARGATE --config-name tutorial
   ```

2. Create a CLI profile using your access key and secret key:

   ```bash
   ecs-cli configure profile --access-key AWS_ACCESS_KEY_ID --secret-key AWS_SECRET_ACCESS_KEY --profile-name tutorial
   ```

   **Note**
   If this is the first time that you are configuring the Amazon ECS CLI, these configurations are marked as default. If this is not your first time configuring the Amazon ECS CLI, see the Amazon ECS Command Line Reference in the Amazon Elastic Container Service Developer Guide to set this as the default configuration and profile.

Step 3: Create a Cluster and Security Group

**To create an ECS cluster and security group**

1. Create an Amazon ECS cluster with the `ecs-cli up` command. Because you specified Fargate as your default launch type in the cluster configuration, this command creates an empty cluster and a VPC configured with two public subnets.
Step 4: Create a Compose File

For this step, create a simple Docker compose file that creates a WordPress application. At this time, the Amazon ECS CLI supports Docker compose file syntax versions 1, 2, and 3. This tutorial uses Docker compose v3.

Here is the compose file, which you can name `docker-compose.yml`. The `wordpress` container exposes port 80 for inbound traffic to the web server. It also configures container logs to go to the CloudWatch log group created earlier. This is the recommended best practice for Fargate tasks.

```yaml
version: '3'
services:
  wordpress:
    image: wordpress
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: wordpress
```

In addition to the Docker compose information, there are some parameters specific to Amazon ECS that you must specify for the service. Using the VPC, subnet, and security group IDs from the previous step, create a file named `ecs-params.yml` with the following content:

```yaml
version: 1
task_definition:
  task_execution_role: ecsTaskExecutionRole
  ecs_network_mode: awsvpc
  task_size:
    mem_limit: 0.5GB
    cpu_limit: 256
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - "subnet ID 1"
        - "subnet ID 2"
      security_groups:
```

Note

This command may take a few minutes to complete as your resources are created. Take note of the VPC and subnet IDs that are created as they are used later.

2. Using the AWS CLI, create a security group using the VPC ID from the previous output:

```bash
aws ec2 create-security-group --group-name "my-sg" --description "My security group" --vpc-id "VPC_ID"
```

3. Using AWS CLI, add a security group rule to allow inbound access on port 80:

```bash
aws ec2 authorize-security-group-ingress --group-id "security_group_id" --protocol tcp --port 80 --cidr 0.0.0.0/0
```
Step 5: Deploy the Compose File to a Cluster

After you create the compose file, you can deploy it to your cluster with `ecs-cli compose service up`. By default, the command looks for files called `docker-compose.yml` and `ecs-params.yml` in the current directory; you can specify a different docker compose file with the `--file` option, and a different ECS Params file with the `--ecs-params` option. By default, the resources created by this command have the current directory in their titles, but you can override that with the `--project-name` option. The `--create-log-groups` option creates the CloudWatch log groups for the container logs.

```
ecs-cli compose --project-name tutorial service up --create-log-groups --cluster-config tutorial
```

Step 6: View the Running Containers on a Cluster

After you deploy the compose file, you can view the containers that are running in the service with `ecs-cli compose service ps`.

```
ecs-cli compose --project-name tutorial service ps --cluster-config tutorial
```

Output:

```
WARN[0000] Skipping unsupported YAML option... option name=networks
WARN[0000] Skipping unsupported YAML option for service... option name=networks service name=wordpress
Name                      State       Ports TaskDefinition
a06a6642-12c5-4006-b1d1-033994580605/wordpress  RUNNING  54.146.193.73:80->80/tcp
tutorial:9
```

In the above example, you can see the `wordpress` container from your compose file, and also the IP address and port of the web server. If you point your web browser at that address, you should see the WordPress installation wizard. Also in the output is the `task-id` value for the container. Copy the task ID as you use it in the next step.

Step 7: View the Container Logs

View the logs for the task:

```
ecs-cli logs --task-id a06a6642-12c5-4006-b1d1-033994580605 --follow --cluster-config tutorial
```

**Note**
The `--follow` option tells the Amazon ECS CLI to continuously poll for logs.

Step 8: Scale the Tasks on the Cluster

You can scale up your task count to increase the number of instances of your application with `ecs-cli compose service scale`. In this example, the running count of the application is increased to two.

```
ecs-cli compose --project-name tutorial service scale 2 --cluster-config tutorial
```
Step 9: Clean Up

When you are done with this tutorial, you should clean up your resources so they do not incur any more charges. First, delete the service so that it stops the existing containers and does not try to run any more tasks.

```
ecs-cli compose --project-name tutorial service down --cluster-config tutorial
```

Now, take down your cluster, which cleans up the resources that you created earlier with `ecs-cli up`.

```
ecs-cli down --force --cluster-config tutorial
```

Tutorial: Creating a Cluster with an EC2 Task Using the Amazon ECS CLI

This tutorial shows you how to set up a cluster and deploy a task using the EC2 launch type.

Prerequisites

Complete the following prerequisites:

- Set up an AWS account.
- Install the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 450).
- Install and configure the AWS CLI. For more information, see AWS Command Line Interface.

Step 1: Configure the Amazon ECS CLI

Before you can start this tutorial, you must install and configure the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 450).

The Amazon ECS CLI requires credentials in order to make API requests on your behalf. It can pull credentials from environment variables, an AWS profile, or an Amazon ECS profile. For more information, see Configuring the Amazon ECS CLI (p. 455).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration:
Step 2: Create Your Cluster

The first action you should take is to create a cluster of Amazon ECS container instances that you can launch your containers on with the `ecs-cli up` command. There are many options that you can choose to configure your cluster with this command, but most of them are optional. In this example, you create a simple cluster of two `t2.medium` container instances that use the `id_rsa` key pair for SSH access (substitute your own key pair here).

By default, the security group created for your container instances opens port 80 for inbound traffic. You can use the `--port` option to specify a different port to open, or if you have more complicated security group requirements, you can specify an existing security group to use with the `--security-group` option.

```
ecs-cli up --keypair id_rsa --capability-iam --size 2 --instance-type t2.medium --cluster-config ec2-tutorial
```

This command may take a few minutes to complete as your resources are created. Now that you have a cluster, you can create a Docker compose file and deploy it.

Step 3: Create a Compose File

For this step, create a simple Docker compose file that creates a WordPress application consisting of a web server and a MySQL database. At this time, the Amazon ECS CLI supports Docker compose file syntax versions 1, 2, and 3. Examples for both Docker Compose version 2 and 3 are provided.

The following parameters are supported in Compose files for the Amazon ECS CLI:

- `cap_add` (not valid for tasks using the Fargate launch type)
- `cap_drop` (not valid for tasks using the Fargate launch type)
- `command`
- `cpu_shares`
  
  **Note**
  
  If you're using the Compose version 3 format, `cpu_shares` should be specified in the `ecs-params.yml` file. For more information, see Using Amazon ECS Parameters (p. 546).

- `devices` (not valid for tasks using the Fargate launch type)
- `dns`
- `dns_search`
• entrypoint
• environment: If an environment variable value isn't specified in the Compose file, but it exists in the shell environment, the shell environment variable value is passed to the task definition that is created for any associated tasks or services.

   Important
   We don't recommend using plaintext environment variables for sensitive information, such as credential data.

• env_file

   Important
   We don't recommend using plaintext environment variables for sensitive information, such as credential data.

• extends (Compose file version 1 and 2 only)
• extra_hosts
• healthcheck (Compose file version 3 only)

   Note
   The start_period field isn't supported using the Compose file. To specify a start_period, use the ecs-params.yml file. For more information, see Using Amazon ECS Parameters (p. 546).

• hostname
• image
• labels
• links (not valid for tasks using the Fargate launch type)
• log_driver (Compose file version 1 only)
• log_opt (Compose file version 1 only)
• logging (Compose file version 2 and 3)
   • driver
   • options
• mem_limit (in bytes)

   Note
   If you're using the Compose version 3 format, mem_limit should be specified in the ecs-params.yml file. For more information, see Using Amazon ECS Parameters (p. 546).

• mem_reservation (in bytes)

   Note
   If you're using the Compose version 3 format, mem_reservation should be specified in the ecs-params.yml file. For more information, see Using Amazon ECS Parameters (p. 546).

• ports
• privileged (not valid for tasks using the Fargate launch type)
• read_only
• security_opt
• shm_size (Compose file version 1 and 2 only and not valid for tasks using the Fargate launch type)
• tmpfs (not valid for tasks using the Fargate launch type)
• tty
• ulimits
• user
• volumes
• volumes_from (Compose file version 1 and 2 only)
• working_dir
Important
The build directive isn't supported at this time.

For more information about Docker Compose file syntax, see the Compose file reference in the Docker documentation.

Here is the compose file, which you can call `docker-compose.yml`. Each container has 100 CPU units and 500 MiB of memory. The `wordpress` container exposes port 80 to the container instance for inbound traffic to the web server. A logging configuration for the containers is also defined.

Example 1: Docker Compose version 2

```yaml
version: '2'
services:
  wordpress:
    image: wordpress
    cpu_shares: 100
    mem_limit: 524288000
    ports:
      - "80:80"
    links:
      - mysql
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial-wordpress
        awslogs-region: us-east-1
        awslogs-stream-prefix: wordpress
  mysql:
    image: mysql:5.7
    cpu_shares: 100
    mem_limit: 524288000
    environment:
      MYSQL_ROOT_PASSWORD: password
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial-mysql
        awslogs-region: us-east-1
        awslogs-stream-prefix: mysql
```

Example 2: Docker Compose version 3

```yaml
version: '3'
services:
  wordpress:
    image: wordpress
    ports:
      - "80:80"
    links:
      - mysql
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial-wordpress
        awslogs-region: us-east-1
        awslogs-stream-prefix: wordpress
  mysql:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
    logging:
      driver: awslogs
```
Step 4: Deploy the Compose File to a Cluster

After you create the compose file, you can deploy it to your cluster with the `ecs-cli compose up` command. By default, the command looks for a compose file called `docker-compose.yml` and an optional ECS parameters file called `ecs-params.yml` in the current directory, but you can specify a different file with the `--file` option. By default, the resources created by this command have the current directory in the title, but you can override that with the `--project-name` option. The `--create-log-groups` option creates the CloudWatch log groups for the container logs.

```
ecs-cli compose up --create-log-groups --cluster-config ec2-tutorial
```

Step 5: View the Running Containers on a Cluster

After you deploy the compose file, you can view the containers that are running on your cluster with the `ecs-cli ps` command.

```
ecs-cli ps
```

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskDefinition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>340488e0-a307-4322-b41c-99f1b70e97f9/wordpress</td>
<td>RUNNING</td>
<td>52.89.204.137:80-&gt;80/tcp</td>
</tr>
<tr>
<td>ecscompose-docker-compose:2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>340488e0-a307-4322-b41c-99f1b70e97f9/mysql</td>
<td>RUNNING</td>
<td></td>
</tr>
<tr>
<td>ecscompose-docker-compose:2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the above example, you can see the `wordpress` and `mysql` containers from your compose file, and also the IP address and port of the web server. If you point a web browser to that address, you should see the WordPress installation wizard.

Step 6: Scale the Tasks on a Cluster

You can scale your task count up so you could have more instances of your application with the `ecs-cli compose scale` command. In this example, you can increase the count of your application to two.

```
ecs-cli compose --file hello-world.yml scale 2 --cluster-config ec2-tutorial
```

Now you should see two more containers in your cluster:
Step 7: Create an ECS Service from a Compose File

Now that you know that your containers work properly, you can make sure that they are replaced if they fail or stop. You can do this by creating a service from your compose file with the `ecs-cli compose service up` command. This command creates a task definition from the latest compose file (if it does not already exist) and creates an ECS service with it, with a desired count of 1.

Before starting your service, stop the containers from your compose file with the `ecs-cli compose down` command so that you have an empty cluster to work with.

```
ecs-cli compose --file hello-world.yml down --cluster-config ec2-tutorial
```

Now you can create your service.

```
ecs-cli compose --file hello-world.yml service up --cluster-config ec2-tutorial
```

Output:

```
INFO[0000] Using ECS task definition TaskDefinition=ecscompose-docker-compose:2
INFO[0000] Created an ECS Service serviceName=ecscompose-service-docker-compose taskDefinition=ecscompose-docker-compose:2
INFO[0000] Updated ECS service successfully desiredCount=1 serviceName=ecscompose-service-docker-compose
INFO[0000] Describe ECS Service status desiredCount=1 runningCount=1
INFO[0000] ECS Service has reached a stable state desiredCount=1 runningCount=1
```

Step 8: Clean Up

When you are done with this tutorial, you should clean up your resources so they do not incur any more charges. First, delete the service so that it stops the existing containers and does not try to run any more tasks.

```
ecs-cli compose --file hello-world.yml service rm --cluster-config ec2-tutorial
```

Output:

```
INFO[0000] Updated ECS service successfully desiredCount=0
serviceName=ecscompose-service-docker-compose
```
Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI

This tutorial shows a simple walkthrough of creating an Amazon ECS service that is configured to use service discovery. Many of the service discovery configuration values can be specified with either the ECS parameters file or flags. When flags are used, they take precedence over the ECS parameters file if both are present. When using the Amazon ECS CLI, the compose project name is used as the name for your ECS service.

Prerequisites

It is expected that you have completed the following prerequisites before continuing on:

- Set up an AWS account.
- Install the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 450).

Configure the Amazon ECS CLI

Before you can start this tutorial, you must install and configure the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 450).

The Amazon ECS CLI requires credentials in order to make API requests on your behalf. It can pull credentials from environment variables, an AWS profile, or an Amazon ECS profile. For more information, see Configuring the Amazon ECS CLI (p. 455).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration:

   ```bash
esc-cli configure --cluster ec2-tutorial --region us-east-1 --default-launch-type EC2 --config-name ec2-tutorial
   ```
2. Create a profile using your access key and secret key:

```
ecs-cli configure profile --access-key AWS_ACCESS_KEY_ID --secret-key AWS_SECRET_ACCESS_KEY --profile-name ec2-tutorial
```

**Note**

If this is the first time that you are configuring the Amazon ECS CLI, these configurations are marked as default. If this is not your first time configuring the Amazon ECS CLI, see the Amazon ECS Command Line Reference in the Amazon Elastic Container Service Developer Guide to set this as the default configuration and profile.

---

**Create an Amazon ECS Service Configured to Use Service Discovery**

Use the following steps to create an Amazon ECS service that is configured to use service discovery with the Amazon ECS CLI.

**To create an Amazon ECS service configured to use service discovery**

1. Create an Amazon ECS service named `backend` and create a private DNS namespace named `tutorial` within a VPC. In this example, the task is using the `awsvpc` network mode, so the `container_name` and `container_port` values are not required.

```
ecs-cli compose --project-name backend service up --private-dns-namespace tutorial --vpc vpc-04deee8176dce7d7d --enable-service-discovery
```

**Output:**

```
INFO[0001] Using ECS task definition TaskDefinition="backend:1"
INFO[0002] Waiting for the private DNS namespace to be created...
INFO[0002] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
WARN[0033] Defaulting DNS Type to A because network mode was awsvpc
INFO[0033] Waiting for the Service Discovery Service to be created...
INFO[0065] Created an ECS service service=backend
taskDefinition="backend:1"
INFO[0066] Using existing namespace ns-kvhnzhb5vxplfmls
WARN[0033] Defaulting DNS Type to A because network mode was awsvpc
INFO[0081] (service backend) has started 1 tasks: (task 824b5a76-8f9c-4beb-a64b-6904e320630e). timestamp="2018-09-12 00:00:26 +0000 UTC"
```

2. Create another service named `frontend` in the same private DNS namespace. Because the namespace already exists, the Amazon ECS CLI uses it instead of creating a new one.

```
ecs-cli compose --project-name frontend service up --private-dns-namespace tutorial --vpc vpc-04deee8176dce7d7d --enable-service-discovery
```

**Output:**

```
INFO[0001] Using ECS task definition TaskDefinition="frontend:1"
INFO[0002] Using existing namespace ns-kvhnzhb5vxplfmls
WARN[0033] Defaulting DNS Type to A because network mode was awsvpc
```
3. Verify that the two services are able to discover each other within
the VPC using DNS. The DNS hostname uses the following format:
<service_discovery_service_name>.<service_discovery_namespace>. For this example,
the frontend service can be discovered at frontend.tutorial and the backend service can
be discovered at backend.tutorial. Because these are private DNS namespaces, these DNS names
only resolve when within the specified VPC.

4. To update the service discovery settings, update the settings for the
frontend service. The values
that can be updated are the DNS TTL and the value for the health check custom config failure
threshold.

```
ecs-cli compose --project-name frontend service up --update-service-discovery --dns-type SRV --dns-ttl 120 --healthcheck-custom-config-failure-threshold 2
```

Output:

```
INFO[0001] Using ECS task definition TaskDefinition="frontend:1"
INFO[0001] Updated ECS service successfully desiredCount=1
serviceName=frontend
INFO[0001] Service status desiredCount=1 runningCount=1
serviceName=frontend
INFO[0001] ECS Service has reached a stable state desiredCount=1 runningCount=1
serviceName=frontend
INFO[0001] Waiting for your Service Discovery resources to be updated...
INFO[0002] Cloudformation stack status stackStatus=UPDATE_IN_PROGRESS
```

5. To clean up, delete the Amazon ECS service and the service discovery resources. When the
frontend service is deleted, the Amazon ECS CLI automatically removes the associated service
discovery service.

```
ecs-cli compose --project-name frontend service rm
```

```
INFO[0000] Updated ECS service successfully desiredCount=0
serviceName=frontend
INFO[0001] Service status desiredCount=0 runningCount=1
serviceName=frontend
INFO[0016] Service status desiredCount=0 runningCount=0
serviceName=frontend
INFO[0016] (service frontend) has stopped 1 running tasks: (task 824b5a76-8f9c-4beb-a64b-6904e320630e). timestamp="2018-09-12 00:37:25 +0000 UTC"
INFO[0016] ECS Service has reached a stable state desiredCount=0 runningCount=0
serviceName=frontend
INFO[0016] Deleted ECS service service=frontend
INFO[0016] ECS Service has reached a stable state desiredCount=0 runningCount=0
serviceName=frontend
INFO[0027] Waiting for your Service Discovery Resource to be deleted...
INFO[0027] Cloudformation stack status stackStatus=DELETE_IN_PROGRESS
```
6. To complete the cleanup, delete the `backend` service along with the private DNS namespace that was created with it. The Amazon ECS CLI associates the AWS CloudFormation stack for the private DNS namespace with the Amazon ECS service for which it was created. When the service is deleted, the namespace is also deleted.

```bash
ecs-cli compose --project-name backend service rm --delete-namespace
```

## Amazon ECS Command Line Reference

The following commands are available in the Amazon ECS CLI. Help text for each command is available by appending the `--help` option to the final command argument. List the help text for the Amazon ECS CLI by using the following command:

```bash
ecs-cli --help
```

**Note**

Ensure that you are using the latest version of the Amazon ECS CLI. The latest version is 1.14.0. For release notes, see [Changelog](#).

### Available Commands

- `ecs-cli` (p. 471)
- `ecs-cli configure` (p. 473)
- `ecs-cli up` (p. 479)
- `ecs-cli down` (p. 487)
- `ecs-cli scale` (p. 489)
- `ecs-cli ps` (p. 490)
- `ecs-cli push` (p. 492)
- `ecs-cli pull` (p. 494)
- `ecs-cli images` (p. 496)
- `ecs-cli license` (p. 499)
- `ecs-cli compose` (p. 500)
- `ecs-cli compose service` (p. 511)
- `ecs-cli logs` (p. 535)
- `ecs-cli check-attributes` (p. 537)
- `ecs-cli registry-creds` (p. 539)
- Using Docker Compose File Syntax (p. 545)
- Using Amazon ECS Parameters (p. 546)

### `ecs-cli`

**Description**

The Amazon ECS command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment. The Amazon ECS CLI supports Docker Compose, a popular open-source tool for defining and running multi-container applications.
For a quick walkthrough of the Amazon ECS CLI, see the Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI (p. 458).

Help text is available for each individual subcommand with `ecs-cli subcommand --help`.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

`ecs-cli [--version] [subcommand] [--help]`

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--version</code>, <code>-v</code></td>
<td>Prints the version information for the Amazon ECS CLI. Required: No</td>
</tr>
<tr>
<td><code>--help</code>, <code>-h</code></td>
<td>Show the help text for the specified command. Required: No</td>
</tr>
</tbody>
</table>

**Available Subcommands**

The `ecs-cli` command supports the following subcommands:

**configure**

Configures your AWS credentials, the Region to use, and the ECS cluster name to use with the Amazon ECS CLI. For more information, see `ecs-cli configure` (p. 473).

**migrate**

Migrates a legacy configuration file (ECS CLI v0.6.6 and older) to the new configuration file format (ECS CLI v1.0.0 and later). The command prints a summary of the changes to be made and then asks for confirmation to proceed. For more information, see `ecs-cli configure migrate` (p. 479).

**up**

Creates the ECS cluster (if it does not already exist) and the AWS resources required to set up the cluster. For more information, see `ecs-cli up` (p. 479).

**down**

Deletes the AWS CloudFormation stack that was created by `ecs-cli up` and the associated resources. For more information, see `ecs-cli down` (p. 487).

**scale**

Modifies the number of container instances in an ECS cluster. For more information, see `ecs-cli scale` (p. 489).

**logs**

Retrieves container logs from CloudWatch Logs. Only valid for tasks that use the `awslogs` driver and has a log stream prefix specified. For more information, see `ecs-cli logs` (p. 535).
ps

Lists all of the running containers in an ECS cluster. For more information, see ecs-cli ps (p. 490).

push

Pushes an image to an Amazon ECR repository. For more information, see ecs-cli push (p. 492).

pull

Pulls an image from an ECR repository. For more information, see ecs-cli pull (p. 494).

images

Lists all of the running containers in an ECS cluster. For more information, see ecs-cli images (p. 496).

license

Prints the LICENSE files for the Amazon ECS CLI and its dependencies. For more information, see ecs-cli license (p. 499).

compose

Executes docker-compose–style commands on an ECS cluster. For more information, see ecs-cli compose (p. 500).

help

Shows the help text for the specified command.

c Lyons

Configures the AWS Region to use, resource creation prefixes, and the Amazon ECS cluster name to use with the Amazon ECS CLI. Stores a single named cluster configuration in the ~/.ecs/config file. The first cluster configuration that is created is set as the default.

Important
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

Working with Multiple Cluster Configurations

The following should be noted when using multiple cluster configurations:

- Multiple cluster configurations may be stored, but one is always the default.
- The first cluster configuration that is stored is set as the default.
- Use the ecs-cli configure default command to change which cluster configuration is set as the default. For more information, see ecs-cli configure default (p. 475).
- A non-default cluster configuration can be referenced in a command by using the --cluster-config flag.

For more information, see ecs-cli configure default (p. 475).

Note
Ensure that you are using the latest version of the Amazon ECS CLI to use all configuration options.
Syntax

`ecs-cli configure --cluster cluster_name --region region [--config-name config_name] [--cfn-stack-name stack_name] [--default-launch-type launch_type] [--help]`

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--cluster, -c cluster_name</code></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: Yes</td>
</tr>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the Region configured using either the <code>ecs-cli configure</code> or <code>aws configure</code> commands.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: Yes</td>
</tr>
<tr>
<td><code>--config-name config_name</code></td>
<td>Specifies the name of this cluster configuration. This is the name that can be referenced in commands using the <code>--cluster-config</code> flag. If this option is omitted, then the name is set to <code>default</code>.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cfn-stack-name stack_name</code></td>
<td>Specifies the stack name to add to the AWS CloudFormation stack that is created on <code>ecs-cli up</code>.</td>
</tr>
<tr>
<td></td>
<td><strong>Important</strong></td>
</tr>
<tr>
<td></td>
<td>It is not recommended to use this parameter. It is included to ensure backwards compatibility with previous versions of the ECS CLI.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Default: <code>amazon-ecs-cli-setup-&lt;cluster_name&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--default-launch-type launch_type</code></td>
<td>Specifies the default launch type to use. Valid values are <code>FARGATE</code> or <code>EC2</code>. If not specified, no default launch type is used. For more information about launch types, see <code>Amazon ECS Launch Types (p. 227)</code>.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--help, -h</code></td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>
Examples

Example

This example configures the Amazon ECS CLI to create a cluster configuration named `ecs-cli-demo`, which uses FARGATE as the default launch type for cluster `ecs-cli-demo` in the `us-east-1` region.

```
ecs-cli configure --region us-east-1 --cluster ecs-cli-demo --default-launch-type FARGATE --config-name ecs-cli-demo
```

Output:

```
INFO[0000] Saved ECS CLI cluster configuration ecs-cli-demo.
```

Contents of the `~/.ecs/config` file after running the command:

```
version: v1
default: fargate
clusters:
  ecs-cli-demo:
    cluster: ecs-cli-demo
    region: us-east-1
    default_launch_type: "FARGATE"
```

**ecs-cli configure default**

Sets the cluster configuration to be read from by default.

**Note**

Unlike the AWS CLI, the Amazon ECS CLI does not expect or require that the default configuration be named `default`. The name of a configuration does not determine whether it is default.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```
ecs-cli configure default --config-name config_name [--help]
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--config-name <code>config_name</code></td>
<td>Specifies the name of the cluster configuration to use by default in subsequent commands. Type: String Required: Yes</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
</tbody>
</table>
### Examples

**Example**

This example configures the Amazon ECS CLI to set the `ecs-cli-demo` cluster configuration as the default.

```
ecs-cli configure default --config-name ecs-cli-demo
```

There is no output if the command is successful.

---

### ecs-cli configure profile

Configures your AWS credentials in a named Amazon ECS profile, which is stored in the `~/.ecs/credentials` file. If multiple profiles are created, you can change the profile used by default with the `ecs-cli configure profile default` command. For more information, see *ecs-cli configure profile default* (p. 478).

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see *Installing the Amazon ECS CLI* (p. 450).

You can configure your AWS credentials in several ways:

- You can set the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables. When you run `ecs-cli configure profile`, the values of those variables are stored in the Amazon ECS CLI configuration file.
- You can pass credentials directly on the command line with the `--access-key`, `--secret-key`, and `--session-token` options.
- You can provide the name of a new profile with the `--profile-name` flag. If a profile name is not provided, then the profile is named `default`.
- The first profile configured is set as the default profile. The Amazon ECS CLI uses credentials specified in this profile unless the `--ecs-profile` flag is used.

---

### Working with Multiple Profiles

The following should be noted when using multiple profiles:

- Multiple profiles may be configured, but one is always the default. This profile is used when an Amazon ECS CLI command is run that requires credentials.
- The first profile that is created is set as the default profile.
- To change the default profile, use the `ecs-cli configure profile default` command. For more information, see *ecs-cli configure profile default* (p. 478).
- A non-default profile can be referenced in a command using the `--ecs-profile` flag.

---

### Syntax

```
ecs-cli configure profile --profile-name profile_name --access-key aws_access_key_id --secret-key aws_secret_access_key [--session-token token] [--help]
```
### Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--profile-name <code>profile_name</code></td>
<td>Specifies the name of this ECS profile. This is the name that can be referenced in commands using the <code>--ecs-profile</code> flag. If this option is omitted, then the name is set to <code>default</code>.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: Yes</td>
</tr>
<tr>
<td>--access-key <code>aws_access_key_id</code></td>
<td>Specifies the AWS access key to use. If the <code>AWS_ACCESS_KEY_ID</code> environment variable is set when <code>ecs-cli configure profile</code> is run, then the AWS access key ID is set to the value of that environment variable.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: Yes</td>
</tr>
<tr>
<td>--secret-key <code>aws_secret_access_key</code></td>
<td>Specifies the AWS secret key to use. If the <code>AWS_SECRET_ACCESS_KEY</code> environment variable is set when <code>ecs-cli configure profile</code> is run, then the AWS secret access key is set to the value of that environment variable.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: Yes</td>
</tr>
<tr>
<td>--session-token <code>token</code></td>
<td>Specifies the AWS session token to use. If the <code>AWS_SESSION_TOKEN</code> environment variable if it is set when <code>ecs-cli configure profile</code> is run, then the AWS session token is set to the value of that environment variable. For more information about using a session token for temporary access, see Requesting Temporary Security Credentials.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

### Examples

#### Example 1

This example configures the Amazon ECS CLI to create and use a profile named `default` with a set of access keys.

```
ecs-cli configure profile --profile-name default --access-key $AWS_ACCESS_KEY_ID --secret-key $AWS_SECRET_ACCESS_KEY
```

Output:
Example 2

This example configures the Amazon ECS CLI to create and use a profile named `default` with a set of access keys and an AWS session token.

```
ecs-cli configure profile --profile-name default --access-key $AWS_ACCESS_KEY_ID --secret-key $AWS_SECRET_ACCESS_KEY --session-token $AWS_SESSION_TOKEN
```

Output:

```
INFO[0000] Saved ECS CLI profile configuration default.
```

**ecs-cli configure profile default**

Sets the Amazon ECS profile to be read from by default.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```
ecs-cli configure profile default --profile-name profile_name [--help]
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--profile-name profile_name</td>
<td>Specifies the name of the ECS profile to be marked as default.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: Yes</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

**Examples**

**Example**

This example configures the Amazon ECS CLI to set the `default` profile as the default profile to be used.

```
ecs-cli configure profile default --profile-name default
```

There is no output if the command is successful.
ecs-cli configure migrate

Migrates a legacy configuration file (ECS CLI v0.6.6 and older) to the new configuration file format (ECS CLI v1.0.0 and later). The command prints a summary of the changes to be made and then asks for confirmation to proceed.

Important
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

Syntax

ecs-cli configure migrate [--force] [--help]

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--force</td>
<td>Omits the interactive description and confirmation step that normally occurs during the configuration file migration. Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command. Required: No</td>
</tr>
</tbody>
</table>

Examples

Example

This example migrates the legacy Amazon ECS CLI configuration file to the new YAML format.

ecs-cli configure migrate

ecs-cli up

Creates the Amazon ECS cluster (if it does not already exist) and the AWS resources required to set up the cluster.

This command creates a new AWS CloudFormation stack called amazon-ecs-cli-setup-cluster_name. You can view the progress of the stack creation in the AWS Management Console.

Important
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

Syntax

ecs-cli up [--capability-iam | --instance-role instance-profile-name] [--keypair keypair_name] [--size n] [--azs availability_zone_1,availability_zone_2] [--security-group security_group_id[,security_group_id[,...]]] [--cidr ip_range] [--port port_number] [--subnets subnet_1,subnet_2] [--vpc vpc_id] [--extra-user-
```
```

## Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--capability-iam</code></td>
<td>Acknowledges that this command may create IAM resources.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This parameter is only supported with tasks that use the EC2 launch type.</td>
</tr>
<tr>
<td></td>
<td>This parameter is required if you do not specify an instance profile name with ```--instance-role``. You cannot specify both options.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--keypair keypair_name</code></td>
<td>Specifies the name of an existing Amazon EC2 key pair to enable SSH access to the EC2 instances in your cluster.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This parameter is only supported with tasks that use the EC2 launch type.</td>
</tr>
<tr>
<td></td>
<td>For more information about creating a key pair, see Setting Up with Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--size n</code></td>
<td>Specifies the number of instances to launch and register to the cluster.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This parameter is only supported with tasks that use the EC2 launch type.</td>
</tr>
<tr>
<td></td>
<td>Type: Integer</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--azs availability_zone_1,availability_zone_2</code></td>
<td>Specifies a comma-separated list of two VPC Availability Zones in which to create subnets (these zones must have the available status). We recommend this option if you do not specify a VPC ID with the <code>--vpc</code> option.</td>
</tr>
<tr>
<td></td>
<td><strong>Warning</strong></td>
</tr>
<tr>
<td></td>
<td>Leaving this option blank can result in a failure to launch container instances when the randomly chosen zone is unavailable.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
</tbody>
</table>
## Name | Description
--- | ---
--security-group | Specifies a comma-separated list of existing security groups to associate with your container instances. If you do not specify a security group here, then a new one is created. For more information, see Security Groups in the Amazon EC2 User Guide for Linux Instances. Required: No

--- security_group_id[,] | Specifies a comma-separated list of existing security groups to associate with your container instances. If you do not specify a security group here, then a new one is created. For more information, see Security Groups in the Amazon EC2 User Guide for Linux Instances. Required: No

--cidr ip_range | Specifies a CIDR/IP range for the security group to use for container instances in your cluster.

**Note**
This parameter is ignored if an existing security group is specified with the --security-group option.

Type: CIDR/IP range
Default: 0.0.0.0/0
Required: No

--port port_number | Specifies a port to open on the security group to use for container instances in your cluster.

**Note**
This parameter is ignored if an existing security group is specified with the --security-group option.

Type: Integer
Default: 80
Required: No

--subnets subnet_1,subnet_2 | Specifies a comma-separated list of existing VPC subnet IDs in which to launch your container instances.

Type: String
Required: This option is required if you specify a VPC with the --vpc option.

--vpc vpc_id | Specifies the ID of an existing VPC in which to launch your container instances. If you specify a VPC ID, you must specify a list of existing subnets in that VPC with the --subnets option. If you do not specify a VPC ID, a new VPC is created with two subnets.

Type: String
Required: No
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--extra-user-data</td>
<td>Specifies additional user data for your container instance. Files can be shell scripts or cloud-init directives. They are packaged into a MIME multipart archive along with user data provided by the Amazon ECS CLI that directs instances to join your cluster. For more information, see Specifying User Data (p. 484).</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--instance-type</td>
<td>Specifies the Amazon EC2 instance type for your container instances. If you specify an A1 instance type, for example a1.medium, and omit the --image-id parameter, the ECS CLI uses the the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI AMI ID for the container instance.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This parameter is supported only with tasks that use the EC2 launch type.</td>
</tr>
<tr>
<td></td>
<td>For more information on EC2 instance types, see Amazon EC2 Instances.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Default: t2.micro</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--image-id</td>
<td>Specifies the Amazon EC2 AMI ID to use for your container instances.</td>
</tr>
<tr>
<td></td>
<td>If you don't specify an AMI ID, the Amazon ECS CLI automatically retrieves the latest stable Amazon ECS-optimized Amazon Linux 2 AMI by querying the SSM Parameter Store API during the cluster resource creation process. This requires the user account that you're using to have the required SSM permissions. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 91).</td>
</tr>
<tr>
<td></td>
<td>If you specify an A1 instance type for the --instance-type parameter and omit the --image-id parameter, the ECS CLI uses the the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI AMI ID for the container instance.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>This parameter is supported only with tasks that use the EC2 launch type.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Default: The latest stable Amazon ECS–optimized AMI for the specified Region.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--no-associate-public-ip-address</td>
<td>Do not assign public IP addresses to new instances in this VPC. Unless this option is specified, new instances in this VPC receive an automatically assigned public IP address. Note This parameter is only supported with tasks that use the EC2 launch type. Required: No</td>
</tr>
<tr>
<td>--force, -f</td>
<td>Forces the recreation of any existing resources that match your current configuration. This option is useful for cleaning up stale resources from previous failed attempts. Required: No</td>
</tr>
<tr>
<td>--tags key1=value1,key2=value2</td>
<td>Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: key1=value1,key2=value2,key3=value3. For more information, see Tagging Resources (p. 485). Type: Key value pairs Required: No</td>
</tr>
<tr>
<td>--instance-role, -f instance-profile-name</td>
<td>Specifies a custom IAM instance profile name for instances in your cluster. Note This parameter is only supported with tasks that use the EC2 launch type. This parameter is required if you do not specify the --capability-iam option. You cannot specify both options. Required: No</td>
</tr>
<tr>
<td>--launch-type launch_type</td>
<td>Specifies the launch type to use. Available options are FARGATE or EC2. For more information about launch types, see Amazon ECS Launch Types (p. 227). This overrides the default launch type stored in your cluster configuration. Type: String Required: No</td>
</tr>
<tr>
<td>--verbose, --debug</td>
<td>Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No</td>
</tr>
</tbody>
</table>

Note: This parameter is only supported with tasks that use the EC2 launch type.
## Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <em>configure</em> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cluster-config cluster_config_name</code></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--ecs-profile ecs_profile</code></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <em>configure profile</em> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--aws-profile aws_profile</code></td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cluster, -c cluster_name</code></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <em>configure</em> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--empty, -e</code></td>
<td>Specifies that an ECS cluster is created with no resources. If other flags are also specified that would create resources, they are ignored and a warning is displayed.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--help, -h</code></td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

## Specifying User Data

When launching tasks that use the EC2 launch type, the ECS CLI always creates container instances that include the following user data:

```bash
#!/bin/bash
echo ECS_CLUSTER={ clusterName } >> /etc/ecs/ecs.config
```
This user data directs the container instance to join your ECS cluster. You can optionally include additional user data using the --extra-user-data flag. The flag can be specified multiple times. For example, extra user data can be shell scripts or cloud-init directives. For more information, see Running Commands on Your Linux Instance at Launch in the Amazon EC2 User Guide for Linux Instances.

The Amazon ECS CLI takes the user data and packs it into a MIME multipart archive, which can be used by cloud-init on the container instance. The Amazon ECS CLI allows existing MIME multipart archives to be passed in with --extra-user-data. The Amazon ECS CLI unpacks the existing archive, and then repack it into the final archive (preserving all header and content type information). The following is an example:

```
ecs-cli up \
    --capability-iam \
    --extra-user-data my-shellscrip \
    --extra-user-data my-cloud-boot-hook \
    --extra-user-data my-mime-multipart-archive \
    --launch-type EC2
```

### Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see AWS Tagging Strategies.

If you specify resource tags when using the `ecs-cli up` command, the Amazon ECS cluster as well as the following resources created by the AWS CloudFormation stack can be tagged:

- Container instances
  
  **Note**
  In order for your container instances to allow tags, you need to opt in to the new Amazon ECS resource ARN formats. For more information, see Amazon Resource Names (ARNs) and IDs (p. 355).

- VPC
- Subnets
- Internet gateway
- Route tables
- Security group
- Autoscaling group

**Note**
For the autoscaling group, the ECS CLI adds a Name tag whose value is the ECS Instance - <CloudFormation stack name>, which is propagated to your container instances. You can override this behavior by specifying your own Name tag.

### Examples

#### Creating a Cluster for Tasks Using the EC2 Launch Type

This example brings up a cluster of four `c4.large` container instances and configures them to use the EC2 key pair called `id_rsa`.

```
ecs-cli up --keypair id_rsa --capability-iam --size 4 --instance-type c4.large --launch-type EC2
```
Creating a Cluster with Container Instances That Use the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI

This example brings up a cluster of one `a1.medium` container instances which will use the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI.

```
ecs-cli up --capability-iam --instance-type a1.medium --launch-type EC2 --region us-east-2
```

Output:

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.
INFO[0000] Using Arm ecs-optimized AMI because instance type was a1.medium
INFO[0001] Using recommended Amazon Linux 2 AMI with ECS Agent 1.25.3 and Docker version 18.06.1-ce
INFO[0000] Created cluster       cluster=ecs-cli-ec2-demo
INFO[0000] Waiting for your cluster resources to be created
INFO[0001] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0061] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0121] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0181] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
Cluster creation succeeded.
VPC created: vpc-abcd1234
Security Group created: sg-abcd1234
Subnets created: subnet-abcd1234
Subnets created: subnet-dcba4321
```

Creating a Cluster for Tasks Using the Fargate Launch Type

This example brings up a cluster for your Fargate tasks and creates a new VPC with two subnets.

```
ecs-cli up --launch-type FARGATE
```

Output:

```
INFO[0001] Created cluster       cluster=ecs-cli-fargate-demo
region=us-west-2
INFO[0000] Waiting for your cluster resources to be created...
INFO[0003] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0003] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0000] Waiting for your cluster resources to be created...
INFO[0066] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0066] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-abcd1234
```
Creating an Empty Cluster

This example brings up an empty cluster named `ecs-cli-empty-demo` with no resources.

```bash
esc-cli up --empty --cluster ecs-cli-empty-demo
```

Output:

```
INFO[0000] Created cluster
region=us-east-1
cluster=ecs-cli-empty-demo
Cluster creation succeeded.
```

`ecs-cli down`

Deletes the AWS CloudFormation stack that was created by `ecs-cli up` and the associated resources.

**Note**
The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren’t created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

The `ecs-cli down` command attempts to delete the cluster specified in `~/.ecs/config`. However, if there are any active services (even with a desired count of 0) or registered container instances in your cluster that were not created by `ecs-cli up`, the cluster is not deleted and the services and pre-existing container instances remain active. This might happen, for example, if you used an existing ECS cluster with registered container instances, such as the default cluster.

If you have remaining services or container instances in your cluster that you would like to remove, you can follow the procedures in Cleaning Up your Amazon ECS Resources (p. 27) to remove them and then delete your cluster.

**Important**
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

`ecs-cli down [--force] [--cluster cluster_name] [--region region] [--help]`

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--force, -f</td>
<td>Acknowledges that this command permanently deletes resources and bypasses the confirmation prompt. Required: No</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cluster-config</code></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td><code>cluster_config_name</code></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--ecs-profile</code> <code>ecs_profile</code></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--aws-profile</code> <code>aws_profile</code></td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cluster, -c cluster_name</code></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--help, -h</code></td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

**Examples**

**Example 1**

This example deletes a cluster that contains resources.

```
ecs-cli down --cluster ecs-cli-fargate-demo --force
```

Output:

```
INFO[0001] Waiting for your cluster resources to be deleted
INFO[0001] Cloudformation stack status stackStatus=DELETE_IN_PROGRESS
INFO[0062] Cloudformation stack status stackStatus=DELETE_IN_PROGRESS
INFO[0123] Cloudformation stack status stackStatus=DELETE_IN_PROGRESS
```
Example 2

This example deletes an empty cluster.

```
ecs-cli down --cluster ecs-cli-empty-demo --force
```

Output:

```
INFO[0002] No CloudFormation stack found for cluster 'ecs-cli-empty-demo'.
INFO[0003] Deleted cluster cluster=ecs-cli-empty-demo
```

es-cli scale

Modifies the number of container instances in your cluster. This command changes the desired and maximum instance count in the Auto Scaling group created by the `ecs-cli up` command. You can use this command to scale out (increase the number of instances) or scale in (decrease the number of instances) your cluster.

**Note**
The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren't created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

**Important**
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```
escs-cli scale --capability-iam --size n [--cluster cluster_name] [--region region] [--help]
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--capability-iam</td>
<td>Acknowledges that this command may create IAM resources. Required: Yes</td>
</tr>
<tr>
<td>--size n</td>
<td>Specifies the number of instances to maintain in your cluster. Type: Integer Required: Yes</td>
</tr>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No</td>
</tr>
</tbody>
</table>
### Examples

#### Example

This example scales the current cluster to two container instances.

```bash
ecs-cli scale --size 2 --capability-iam
```

**Output:**

```
INFO[0001] Waiting for your cluster resources to be updated
INFO[0001] Cloudformation stack status stackStatus=UPDATE_IN_PROGRESS
```

### ecs-cli ps

Lists all running containers in your Amazon ECS cluster.

The IP address displayed by the Amazon ECS CLI depends heavily upon how you have configured your task and cluster:
• For tasks using the EC2 launch type without task networking, the IP address shown is the public IP address of the Amazon EC2 instance running your task, or the instance private IP address if it lacks a public IP address.

• For tasks using the EC2 launch type with task networking, the ECS CLI only shows a private IP address obtained from the network interfaces section of the Describe Task output for the task.

• For tasks using the Fargate launch type, the Amazon ECS CLI returns the public IP address assigned to the elastic network instance attached to the Fargate task. If the elastic network instance lacks a public IP address, then the Amazon ECS CLI falls back to the private IP address obtained from the network interfaces section of the Describe Task output.

Syntax

ecs-cli ps [--desired-status status] [--cluster cluster_name] [--region region] [--help]

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--desired-status</td>
<td>The container desired status to filter the container list results with.</td>
</tr>
<tr>
<td>status</td>
<td>Required: No</td>
</tr>
<tr>
<td></td>
<td>Valid values: RUNNING</td>
</tr>
<tr>
<td>--region, -r</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</td>
</tr>
<tr>
<td>region</td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster-config</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td>cluster_config_name</td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--ecs-profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</td>
</tr>
<tr>
<td>ecs_profile</td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--aws-profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials.</td>
</tr>
<tr>
<td>aws_profile</td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>
**Name**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cluster, -c cluster_name</td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <strong>configure</strong> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

**Examples**

**Example**

This example shows the containers that are running in the cluster.

```
ecs-cli ps
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Health</th>
<th>State</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskDefinition</td>
<td>Health</td>
<td>State</td>
<td>Ports</td>
</tr>
<tr>
<td>afd7f8a0-3813-4e1a-9d9e-ca7e9d1fcfbb/wordpress</td>
<td>HEALTHY</td>
<td>RUNNING</td>
<td>36.253.177.221:80-&gt;80/tcp</td>
</tr>
<tr>
<td>compose3:7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dca67e02-68ca-4507-b194-a47239b5e7a9/wordpress</td>
<td>UNKNOW</td>
<td>RUNNING</td>
<td>37.234.146.14:80-&gt;80/tcp</td>
</tr>
<tr>
<td>healthcheck:3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dca67e02-68ca-4507-b194-a47239b5e7a9/redis</td>
<td>HEALTHY</td>
<td>RUNNING</td>
<td></td>
</tr>
<tr>
<td>healthcheck:3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>feb6e10e-3385-4c9b-a6cb-787cc8e90dda/sample-app</td>
<td>UNKNOW</td>
<td>RUNNING</td>
<td>54.229.211.206:80-&gt;80/tcp</td>
</tr>
<tr>
<td>tutorial-task-def:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tutorial-task-def:1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ecs-cli push**

Pushes an image to an Amazon ECR repository.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, seeInstalling the Amazon ECS CLI (p. 450).

**Syntax**

```
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--registry-id</td>
<td>Specifies the Amazon ECR registry ID to which to push the image. By default, images are pushed to the current AWS account.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--tags value</td>
<td>Specifies the metadata to apply to your Amazon ECR repository. Each tag consists of a key and an optional value. Tag keys can have a maximum character length of 128 characters, and tag values can have a maximum length of 256 characters. Tags use the following format: key1=value1,key2=value2,key3=value3.</td>
</tr>
<tr>
<td></td>
<td>Type: Key value pairs</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--verbose, --debug</td>
<td>Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--use-fips</td>
<td>Routes calls to Amazon ECR through FIPS endpoints.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster-config cluster_config_name</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--ecs-profile ecs_profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--aws-profile aws_profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>
### Name and Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cluster, -c <em>cluster_name</em></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command. Type: String Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command. Required: No</td>
</tr>
</tbody>
</table>

### Using FIPS Endpoints

The Amazon ECS CLI supports using FIPS endpoints for calls to Amazon ECR. To ensure that you're accessing Amazon ECR using FIPS endpoints, use the `--use-fips` flag on the `push`, `pull`, or `images` command. FIPS endpoints are currently available in us-west-1, us-west-2, us-east-1, us-east-2, and AWS GovCloud (US). For more information, see Federal Information Processing Standard (FIPS) 140-2.

### Examples

#### Example 1

This example pushes a local image called *ubuntu* to an Amazon ECR repository with the same name.

```bash
esc-cli push ubuntu
```

Output:

```
INFO[0000] Getting AWS account ID...
INFO[0000] Tagging image
  repository="aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu" source-image=ubuntu
tag=
INFO[0000] Image tagged
INFO[0001] Creating repository repository=ubuntu
INFO[0001] Repository created
INFO[0001] Pushing image
  repository="aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu" tag=
INFO[0079] Image pushed
```

#### ecs-cli pull

Pull an image from an Amazon ECR repository.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

### Syntax

```
esc-cli pull [--registry-id *registry_id*] [--region *region*] [--verbose] [--use-fips] ECR_REPOSITORY[:TAG|@DIGEST] [--help]
```
# Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--registry-id registry_id</code></td>
<td>Specifies the Amazon ECR registry ID from which to pull the image. By default, images are pulled from the current AWS account. Required: No</td>
</tr>
<tr>
<td><code>--verbose, --debug</code></td>
<td>Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No</td>
</tr>
<tr>
<td><code>--use-fips</code></td>
<td>Routes calls to Amazon ECR through FIPS endpoints. Required: No</td>
</tr>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command. Required: No</td>
</tr>
<tr>
<td><code>--cluster-config cluster_config_name</code></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Required: No</td>
</tr>
<tr>
<td><code>--ecs-profile ecs_profile</code></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command. Required: No</td>
</tr>
<tr>
<td><code>--aws-profile aws_profile</code></td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/aws/credentials</code>. Required: No</td>
</tr>
<tr>
<td><code>--cluster, -c cluster_name</code></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command. Required: No</td>
</tr>
<tr>
<td><code>--help, -h</code></td>
<td>Shows the help text for the specified command. Required: No</td>
</tr>
</tbody>
</table>
Using FIPS Endpoints

The Amazon ECS CLI supports using FIPS endpoints for calls to Amazon ECR. To ensure that you’re accessing Amazon ECR using FIPS endpoints, use the `--use-fips` flag on the push, pull, or images command. FIPS endpoints are currently available in us-west-1, us-west-2, us-east-1, us-east-2, and AWS GovCloud (US). For more information, see Federal Information Processing Standard (FIPS) 140-2.

Examples

Example 1

This example pulls a local image called `amazonlinux` from an Amazon ECR repository with the same name.

```bash
ecs-cli pull amazonlinux
```

Output:

```
INFO[0000] Getting AWS account ID...
INFO[0000] Pulling image
 repository="aws_account_id.dkr.ecr.us-east-1.amazonaws.com/amazonlinux" tag=
INFO[0129] Image pulled
```

ecs-cli images

List images in an Amazon ECR registry or repository.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

Syntax

```
```

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--registry-id registry_id</code></td>
<td>Specifies the Amazon ECR registry with which to list images. By default, images are listed for the current AWS account. Required: No</td>
</tr>
<tr>
<td><code>--tagged</code></td>
<td>Filters the result to show only tagged images. Required: No</td>
</tr>
<tr>
<td><code>--untagged</code></td>
<td>Filters the result to show only untagged images. Required: No</td>
</tr>
</tbody>
</table>
### Using FIPS Endpoints

The Amazon ECS CLI supports using FIPS endpoints for calls to Amazon ECR. To ensure you are accessing Amazon ECR using FIPS endpoints, use the `--use-fips` flag on the `push`, `pull`, or `images` command. FIPS endpoints are currently available in `us-west-1`, `us-west-2`, `us-east-1`, `us-east-2`, and AWS GovCloud. For more information, see [Federal Information Processing Standard (FIPS) 140-2](https://csrc.nist.gov/publications/fips/fips140-2/fips140-2.pdf).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Required: No</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--verbose</code>, <code>--debug</code></td>
<td>Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues.</td>
<td>No</td>
</tr>
<tr>
<td><code>--use-fips</code></td>
<td>Routes calls to Amazon ECR through FIPS endpoints.</td>
<td>No</td>
</tr>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
<td>String</td>
</tr>
<tr>
<td><code>--cluster-config</code></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
<td>String</td>
</tr>
<tr>
<td><code>--ecs-profile</code></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure</code> command.</td>
<td>String</td>
</tr>
<tr>
<td><code>--aws-profile</code></td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</td>
<td>String</td>
</tr>
<tr>
<td><code>--cluster, -c cluster_name</code></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
<td>String</td>
</tr>
<tr>
<td><code>--help, -h</code></td>
<td>Shows the help text for the specified command.</td>
<td>No</td>
</tr>
</tbody>
</table>
**Examples**

**Example 1**

This example lists all of the images in an Amazon ECR registry.

```
ecs-cli images
```

Output:

```
REPOSITORY NAME  TAG                 IMAGE DIGEST
PUSHED AT           SIZE
rkt              latest
sha256:404758ad8af94347fc8582fc8e30b6284f2b0751de39b2e755da212f80232fac  3 months ago
203 MB
foobuntu          latest
sha256:6b079ae764a6affcb632231349da5e1b084bece8c46883c099863ee2aeb5cf8  4 days ago
51.7 MB
ubuntu           xenial
sha256:6b079ae764a6affcb632231349da5e1b084bece8c46883c099863ee2aeb5cf8  4 days ago
51.7 MB
ubuntu           latest
sha256:6b079ae764a6affcb632231349da5e1b084bece8c46883c099863ee2aeb5cf8  4 days ago
51.7 MB
ubuntu           <none>
sha256:512e30a26d9fa3648dbccbc9e78e9bab636e6022e2d80bd73c99177b21a0d3982  19 minutes ago
268 MB
ubuntu           trusty
sha256:bd6d24e8fa3f5822146b2c94247976b87e6564195c3c180b67833e6ea699f7c2  18 minutes ago
67.2 MB
ubuntu           precise
sha256:b38267a51fb4460699bc2bcd6b53d42f6c97b4e4f9a819df3b762c3393b2a  17 minutes ago
40.1 MB
amazon-ecs-sample latest
sha256:bf04071a8edec309f4d109ae36f24a5c272a115b6f7ee33577d092405902471c  2 weeks ago
105 MB
golang           latest
sha256:137b22e6ee2df470b0c2de8b2f1e583be0baf09334a5a8b2096193577d983ab  4 days ago
266 MB
amazonlinux      latest
sha256:a59d5635b3139deee8cb108fb97bf3e9021b8ccea6dec8ff49733230cb2f0eca  4 days ago
98.8 MB
awsbatch/fetch_and_run latest
sha256:543800007416d0ccff4f63643bb18e8f4b874eae772128efcd231ff456a37fc  6 weeks ago
116 MB
```

**Example 2**

This example lists all of the images in a specific Amazon ECR repository.

```
ecs-cli images ubuntu
```

Output:

```
REPOSITORY NAME  TAG                 IMAGE DIGEST
PUSHED AT           SIZE
ubuntu           xenial
sha256:6b079ae764a6affcb632231349da5e1b084bece8c46883c099863ee2aeb5cf8  4 days ago
51.7 MB
```
Example 3

This example lists all of the untagged images in an Amazon ECR registry.

```
ecs-cli images --untagged
```

Output:

<table>
<thead>
<tr>
<th>REPOSITORY NAME</th>
<th>TAG</th>
<th>IMAGE DIGEST</th>
<th>PUSHED AT</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu</td>
<td>&lt;none&gt;</td>
<td>sha256:512e30a26d9fa3648dbcccb9e78e9bab636e6022e2d80bd73c99177b21a0d3982</td>
<td>24 minutes ago</td>
<td>268 MB</td>
</tr>
</tbody>
</table>

**ecs-cli license**

Prints the LICENSE files for the Amazon ECS CLI and its dependencies.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```
ecs-cli license [--help]
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command. Required: No</td>
</tr>
</tbody>
</table>

**Examples**

**Example**

This example prints the license files.

```
ecs-cli license
```
ecs-cli compose

Manage Amazon ECS tasks with `docker-compose`-style commands on an ECS cluster. For more information on how Docker Compose file syntax works with the Amazon ECS CLI, see Using Docker Compose File Syntax (p. 545).

**Note**
To create Amazon ECS services with the Amazon ECS CLI, see `ecs-cli compose service` (p. 511).

The `ecs-cli compose` command uses a project name with the task definitions and services it creates. When the CLI creates a task definition from a Compose file, the task definition is called `project-name`. When the CLI creates a service from a Compose file, the service is called `service-project-name`. By default, the project name is the name of the directory that contains your Docker Compose file. However, you can also specify your own project name with the `--project-name` option.

**Note**
The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren't created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

**Important**
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--verbose</code>, <code>--debug</code></td>
<td>Increases the verbosity of command output to aid in diagnostics.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>--file, -f compose_file</code></td>
<td>Specifies the Docker Compose file to use. At this time, the latest version of the Amazon ECS CLI supports Docker Compose file syntax versions 1, 2, and 3. If the <code>COMPOSE_FILE</code> environment variable is set when <code>ecs-cli compose</code> is run, the Docker Compose file is set to the value of that environment variable. Type: String. Default: <code>./docker-compose.yml</code>. Required: No.</td>
</tr>
<tr>
<td><code>--project-name, -p project_name</code></td>
<td>Specifies the project name to use. If the <code>COMPOSE_PROJECT_NAME</code> environment variable is set when <code>ecs-cli compose</code> is run, the project name is set to the value of that environment variable. Type: String. Default: The current directory name. Required: No.</td>
</tr>
<tr>
<td><code>--task-role-arn role_value</code></td>
<td>Specifies the short name or full Amazon Resource Name (ARN) of the IAM role that containers in this task can assume. All containers in this task are granted the permissions that are specified in this role. Type: String. Required: No.</td>
</tr>
<tr>
<td><code>--ecs-params ecs_params_file</code></td>
<td>Specifies the ECS parameters that aren't native to Docker Compose files. For more information, see Using Amazon ECS Parameters (p. 546). Default: <code>./ecs-params.yml</code>. Required: No.</td>
</tr>
<tr>
<td><code>--registry-creds value</code></td>
<td>Specifies the Amazon ECS registry credentials file to use. Defaults to the latest output file from the <code>ecs-cli registry-creds up</code> command, if one exists. For more information, see <code>ecs-cli registry-creds</code> (p. 539). Default: <code>./ecs-registry-creds_[TIMESTAMP].yml</code>. Required: No.</td>
</tr>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command. Type: String. Required: No.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--cluster-config <strong>cluster_config_name</strong></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--ecs-profile <strong>ecs_profile</strong></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <strong>configure profile</strong> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--aws-profile <strong>aws_profile</strong></td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/aws/credentials</code>.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster, -c <strong>cluster_name</strong></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <strong>configure</strong> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

### Available Subcommands

The **ecs-cli compose** command supports the following subcommands. Each of these subcommands has their own flags associated with them, which can be displayed using the **--help** flag.

**create**

Creates an Amazon ECS task definition from your Compose file. For more information, see **ecs-cli compose create** (p. 503).

**ps, list**

Lists all the containers in your cluster that were started by the Compose project.

**run [containerName] ["command ..."]**...

Starts all containers overriding commands with the supplied one-off commands for the containers.

**scale n**

Scales the number of running tasks to the specified count.

**start**

Starts a single task from the task definition created from your Compose file. For more information, see **ecs-cli compose start** (p. 506).
**stop, down**

Stops all the running tasks created by the Compose project.

**up**

Creates an ECS task definition from your Compose file (if it doesn't already exist) and runs one instance of that task on your cluster (a combination of `create` and `start`). For more information, see `ecs-cli compose up (p. 509)`.

**service [subcommand]**

Creates an ECS service from your Compose file. For more information, see `ecs-cli compose service (p. 511)`.

**help**

Shows the help text for the specified command.

### `ecs-cli compose create`

Creates an Amazon ECS task definition from your Compose file.

- **Important**
  - We don't recommend using plaintext environment variables for sensitive information, such as credential data.
  - Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see `Installing the Amazon ECS CLI (p. 450)`.

**Syntax**

```
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cluster-config</code></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td><code>cluster_config_name</code></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--ecs-profile</code></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.</td>
</tr>
<tr>
<td><code>ecs_profile</code></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

API Version 2014-11-13

503
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--aws-profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials. Type: String Required: No</td>
</tr>
<tr>
<td>--cluster, -c</td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No</td>
</tr>
<tr>
<td>--launch-type</td>
<td>Specifies the launch type to use. Available options are FARGATE or EC2. For more information about launch types, see Amazon ECS Launch Types (p. 227). Type: String Required: No</td>
</tr>
<tr>
<td>--create-log-groups</td>
<td>Creates the CloudWatch log groups specified in your Compose files. Required: No</td>
</tr>
</tbody>
</table>
| --tags               | Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format:
key1=value1,key2=value2,key3=value3. For more information, see Tagging Resources (p. 504). Type: Key value pairs Required: No |
| --help, -h           | Shows the help text for the specified command. Required: No                                                                                                                                                  |

**Tagging Resources**

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see AWS Tagging Strategies.

When using the `ecs-cli compose create` command, using the --tags flag enables you to add metadata tags to the task definition.
Examples

Register a Task Definition

This example creates a task definition with the project name hello-world from the hello-world.yml Compose file.

```
ecs-cli compose --project-name hello-world --file hello-world.yml create --launch-type EC2
```

Output:

```
INFO[0000] Using ECS task definition                     TaskDefinition=ecscompose-hello-world:5
```

Register a Task Definition Using the EC2 Launch Type Without Task Networking

This example creates a task definition with the project name hello-world from the hello-world.yml Compose file. Additional ECS parameters specified for the container size parameters.

Example Docker Compose file, named hello-world.yml:

```
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: /ecs/cli/tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named ecs-params.yml:

```
version: 1
task_definition:
  services:
    nginx:
      cpu_shares: 256
      mem_limit: 0.5GB
      mem_reservation: 0.5GB
```

```
ecs-cli compose --project-name hello-world --file hello-world.yml --ecs-params ecs-params.yml --region us-east-1 create --launch-type EC2
```

Output:

```
INFO[0000] Using ECS task definition                     TaskDefinition=ecscompose-hello-world:5
```

Register a Task Definition Using the Fargate Launch Type

This example creates a task definition with the project name hello-world from the hello-world.yml Compose file. Additional ECS parameters are specified for task networking configuration for the Fargate launch type. Then one instance of the task is run.
Example Docker Compose file, named `hello-world.yml`:

```yaml
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named `ecs-params.yml`:

```yaml
version: 1
task_definition:
  task_execution_role: ecsTaskExecutionRole
  ecs_network_mode: awsvpc
  task_size:
    mem_limit: 0.5GB
    cpu_limit: 256
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - subnet-abcd1234
        - subnet-dbca4321
      security_groups:
        - sg-abcd1234
      assign_public_ip: ENABLED
```

Command:

```
ecs-cli compose --project-name hello-world --file hello-world.yml --ecs-params ecs-params.yml --region us-east-1 create --launch-type FARGATE
```

Output:

```
INFO[0000] Using ECS task definition                     TaskDefinition=ecscompose-hello-world:5
```

**ecs-cli compose start**

Starts a single Amazon ECS task from the task definition created from your Compose file.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```
```
# Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cluster-config</code></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td><code>cluster_config_name</code></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--ecs-profile</code></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.</td>
</tr>
<tr>
<td><code>ecs_profile</code></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--aws-profile</code></td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</td>
</tr>
<tr>
<td><code>aws_profile</code></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--cluster, -c cluster_name</code></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--launch-type</code></td>
<td>Specifies the launch type to use. Available options are FARGATE or EC2. For more information about launch types, see Amazon ECS Launch Types (p. 227).</td>
</tr>
<tr>
<td><code>launch_type</code></td>
<td>This overrides the default launch type stored in your cluster configuration.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--create-log-groups</code></td>
<td>Creates the CloudWatch log groups specified in your Compose files.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--help, -h</code></td>
<td>Shows the help text for the specified command.</td>
</tr>
</tbody>
</table>
Examples

Run a Task

This example creates a task definition from the `hello-world.yml` Compose file. Additional ECS parameters are specified for task networking configuration for the Fargate launch type. Then a single task is run using that task definition.

Example Docker Compose file, named `hello-world.yml`:

```yaml
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
    - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named `ecs-params.yml`:

```yaml
version: 1
task_definition:
  task_execution_role: ecsTaskExecutionRole
ecs_network_mode: awsvpc
task_size:
  mem_limit: 0.5GB
cpu_limit: 256
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
      - subnet-abcd1234
      - subnet-dbcd4321
      security_groups:
      - sg-abcd1234
      assign_public_ip: ENABLED
```

Command:

```
ecs-cli compose --file hello-world.yml --ecs-params ecs-params.yml start --launch-type FARGATE --create-log-groups
```

Output:

```
INFO[0000] Using ECS task definition TaskDefinition=ecscompose-hello-world:5
```

API Version 2014-11-13
**ecs-cli compose up**

If an Amazon ECS task definition doesn't already exist, creates one from your Compose file and runs one instance of that task on your cluster.

**Important**
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```bash
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command. Type: String Required: No</td>
</tr>
<tr>
<td>--cluster-config cluster_config_name</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No</td>
</tr>
<tr>
<td>--ecs-profile ecs_profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command. Type: String Required: No</td>
</tr>
<tr>
<td>--aws-profile aws_profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>. Type: String Required: No</td>
</tr>
<tr>
<td>--cluster, -c cluster_name</td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command. Type: String Required: No</td>
</tr>
</tbody>
</table>
## Name
<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--launch-type launch_type</td>
</tr>
<tr>
<td>--create-log-groups</td>
</tr>
<tr>
<td>--force-update</td>
</tr>
<tr>
<td>--tags key1=value1,key2=value2</td>
</tr>
<tr>
<td>--disable-ecs-managed-tags</td>
</tr>
<tr>
<td>--help, -h</td>
</tr>
</tbody>
</table>

### Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see AWS Tagging Strategies.

When using the `ecs-cli compose up` command, using the `--tags` flag enables you to add metadata tags to the task definition and tasks. Amazon ECS managed tags are enabled by default unless specifically disabled using the `--disable-ecs-managed-tags` flag. For more information, see Tagging Your Resources for Billing (p. 361).

### Examples

**Register a Task Definition Using the AWS Fargate Launch Type with Task Networking**

This example creates a task definition with the project name `hello-world` from the `hello-world.yml` Compose file. Additional ECS parameters are specified for task and network configuration for the Fargate launch type. Then one instance of the task is run using the Fargate launch type.
Example Docker Compose file, named hello-world.yml:

```yaml
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named ecs-params.yml:

```yaml
version: 1
task_definition:
  ecs_network_mode: awsvpc
task_execution_role: ecsTaskExecutionRole
task_size:
  cpu_limit: 512
  mem_limit: 2GB
services:
  nginx:
    essential: true
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - subnet-abcd1234
        - subnet-dcba4321
      security_groups:
        - sg-abcd1234
        - sg-dcba4321
    assign_public_ip: ENABLED
```

Command:

```
ecs-cli compose --project-name hello-world --file hello-world.yml --ecs-params ecs-params.yml up --launch-type FARGATE
```

Output:

```
INFO[0000] Using ECS task definition TaskDefinition=ecscompose-hello-world:5
```

**ecs-cli compose service**

Manage Amazon ECS services with `docker-compose`-style commands on an ECS cluster. For more information on how Docker compose file syntax works with the ECS CLI, see Using Docker Compose File Syntax (p. 545).

**Note**
To run tasks with the Amazon ECS CLI instead of creating services, see `ecs-cli compose` (p. 500).

The **ecs-cli compose service** command uses a project name with the task definitions and services that it creates. When the Amazon ECS CLI creates a task definition and service from a compose file, the
task definition and service are called *project-name*. By default, the project name is the name of the directory that contains your Docker compose file. However, you can also specify your own project name with the `--project-name` option.

**Note**
The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren't created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

**Important**
Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```bash
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--verbose, --debug</code></td>
<td>Increases the verbosity of command output to aid in diagnostics.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--file, -f compose_file</code></td>
<td>Specifies the Docker Compose file to use. At this time, the latest version</td>
</tr>
<tr>
<td></td>
<td>of the Amazon ECS CLI supports Docker Compose file syntax versions 1, 2,</td>
</tr>
<tr>
<td></td>
<td>and 3. If the COMPOSE_FILE environment variable is set when `ecs-cli</td>
</tr>
<tr>
<td></td>
<td>compose` is run, the Docker Compose file is set to the value of that</td>
</tr>
<tr>
<td></td>
<td>environment variable.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Default: ./docker-compose.yml</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--project-name, -p</code></td>
<td>Specifies the project name to use. If the COMPOSE_PROJECT_NAME environment</td>
</tr>
<tr>
<td><code>project_name</code></td>
<td>variable is set when <code>ecs-cli compose</code> is run, the project name is set to</td>
</tr>
<tr>
<td></td>
<td>the value of that environment variable.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Default: The current directory name.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td><code>--task-role-arn</code></td>
<td>Specifies the short name or full Amazon Resource Name (ARN) of the IAM</td>
</tr>
<tr>
<td><code>role_value</code></td>
<td>role that contains in this task can assume. All containers in this task are</td>
</tr>
<tr>
<td></td>
<td>granted the permissions that are specified in this role.</td>
</tr>
</tbody>
</table>

API Version 2014-11-13

512
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--ecs-params <em>ecs_params_file</em></td>
<td>Specifies the ECS parameters that aren't native to Docker Compose files. For more information, see Using Amazon ECS Parameters (p. 546). Default: ./ecs-params.yml Required: No</td>
</tr>
<tr>
<td>--registry-creds <em>value</em></td>
<td>Specifies the Amazon ECS registry credentials file to use. Defaults to the latest output file from the <em>ecs-cli registry-creds up</em> command, if one exists. For more information, see <em>ecs-cli registry-creds</em> (p. 539). Default: ./ecs-registry-creds_[TIMESTAMP].yml Required: No</td>
</tr>
<tr>
<td>--region, -r <em>region</em></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No</td>
</tr>
<tr>
<td>--cluster-config <em>cluster_config_name</em></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No</td>
</tr>
<tr>
<td>--ecs-profile <em>ecs_profile</em></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No</td>
</tr>
<tr>
<td>--aws-profile <em>aws_profile</em></td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials. Type: String Required: No</td>
</tr>
<tr>
<td>--cluster, -c <em>cluster_name</em></td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No</td>
</tr>
</tbody>
</table>
Available Subcommands

The `ecs-cli compose service` command supports the following subcommands. Each of these subcommands has its own flags associated with them, which can be displayed using the `--help` flag.

**create**

Creates an Amazon ECS service from your compose file. The service is created with a desired count of 0, so no containers are started by this command. For more information, see `ecs-cli compose service create` (p. 514).

**start**

Starts one copy of each of the containers on the created Amazon ECS service. This command updates the desired count of the service to 1. For more information, see `ecs-cli compose service start` (p. 520).

**up**

Creates an Amazon ECS service from your compose file (if it does not already exist) and runs one instance of that task on your cluster (a combination of `create` and `start`). This command updates the desired count of the service to 1. For more information, see `ecs-cli compose service up` (p. 522).

**ps, list**

Lists all the containers in your cluster that belong to the service created with the compose project. For more information, see `ecs-cli compose service ps, list` (p. 529).

**scale**

Scales the desired count of the service to the specified count. For more information, see `ecs-cli compose service scale` (p. 530).

**stop**

Stops the running tasks that belong to the service created with the compose project. This command updates the desired count of the service to 0. For more information, see `ecs-cli compose service stop` (p. 532).

**rm, delete, down**

Updates the desired count of the service to 0 and then deletes the service. For more information, see `ecs-cli compose service rm, delete, down` (p. 533).

**ecs-cli compose service create**

Creates an Amazon ECS service from your compose file. The service is created with a desired count of 0, so no containers are started by this command.

**Syntax**

```
```

API Version 2014-11-13

514
ecs-cli compose service

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| --deployment-max-percent | Specifies the upper limit (as a percentage of the service's desiredCount) of the number of running tasks that can be running in a service during a deployment. For more information, see `maximumPercent` (p. 295).  
   Default value: 200  
   Required: No |
| --deployment-min-healthy-percent | Specifies the lower limit (as a percentage of the service's desiredCount) of the number of running tasks that must remain running and healthy in a service during a deployment. For more information, see `minimumHealthyPercent` (p. 296).  
   Default value: 100  
   Required: No |
| --target-group-arn | Specifies the full Amazon Resource Name (ARN) of a previously configured Elastic Load Balancing target group to associate with your service.  
   Required: No |
| --container-name | Specifies the container name (as it appears in a container definition). This parameter is required if a load balancer or target group is specified.  
   Required: No, unless a load balancer or target group is specified. |
| --container-port | Specifies the port on the container to associate with the load balancer. This port must correspond to a containerPort in the service's task definition. This parameter is required if a load balancer or target group is specified.  
   Required: No, unless a load balancer or target group is specified. |
| --load-balancer-name | Specifies the name of a previously configured Elastic Load Balancing load balancer to associate with your service.  
   Required: No |
<p>| --role | Specifies the name or full Amazon Resource Name (ARN) of the IAM role that allows Amazon ECS to make calls to your load balancer or target group on your behalf. This parameter is required if you're using a load balancer or target group with your service. If you specify the role parameter, you |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--name</td>
<td>must also specify a load balancer name or target group ARN, along with a container name and container port. Required: No, unless a load balancer or target group is specified.</td>
</tr>
<tr>
<td>--health-check-grace-period</td>
<td>Specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started. Required: No</td>
</tr>
<tr>
<td>--create-log-groups</td>
<td>Creates the CloudWatch log groups specified in your Compose files. Required: No</td>
</tr>
<tr>
<td>--enable-service-discovery</td>
<td>Specifies whether to enable service discovery for this service. Required: No</td>
</tr>
<tr>
<td>--vpc</td>
<td>Specifies the VPC that will be attached to the private DNS namespace for service discovery. This parameter is required if --private-dns-namespace is specified. Required: No</td>
</tr>
<tr>
<td>--private-dns-namespace</td>
<td>Specifies the name of the private DNS namespace to use with service discovery. The Amazon ECS CLI automatically creates the namespace if it doesn't exist. For example, if the namespace is corp, a service named foo is reachable via DNS at foo.corp. If you use this parameter, you must also specify a VPC using the --vpc parameter. Required: No</td>
</tr>
<tr>
<td>--private-dns-name-id</td>
<td>Specifies the ID of an existing private DNS namespace to use with service discovery. If you use this parameter, you can't specify either --private-dns-namespace or --vpc. Required: No</td>
</tr>
<tr>
<td>--public-dns-namespace</td>
<td>Specifies the name of the public DNS namespace to use with service discovery. For example, if the namespace is corp, a service named foo is reachable via DNS at foo.corp. Required: No</td>
</tr>
<tr>
<td>--public-dns-name-id</td>
<td>Specifies the ID of an existing public DNS namespace to use with service discovery. If you use this parameter, you can't specify a --public-dns-namespace. Required: No</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--sd-container-name</td>
<td>Specifies the name of the container, which is referred to as a service in your Docker Compose file. For more information, see <strong>Service configuration reference</strong>. This parameter is required if you're using SRV records. Required: No, unless SRV DNS records are being used.</td>
</tr>
<tr>
<td>--sd-container-port</td>
<td>Specifies the port on the container that will be used for service discovery. This parameter is required if you're using SRV records. Required: No, unless SRV DNS records are being used.</td>
</tr>
<tr>
<td>--dns-ttl</td>
<td>Specifies the amount of time, in seconds, that you want DNS resolvers to cache the settings for the DNS records used for service discovery. Default value: 60 Required: No</td>
</tr>
<tr>
<td>--dns-type</td>
<td>Specifies the type of DNS record used for service discovery. Accepted values are A or SRV. If your task uses either the bridge or host network modes, SRV records are required. If your task uses the awsvpc network mode, A records are the default. Required: No</td>
</tr>
<tr>
<td>--healthcheck-custom-config-failure-threshold</td>
<td>Specifies the number of 30-second intervals that you want the service discovery service to wait after receiving an UpdateInstanceCustomHealthStatus request before it changes the health status. Default value: 1 Required: No</td>
</tr>
</tbody>
</table>
| Name                              | Description                                                                ieszly the scheduling strategy to use for the service. 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 (p. 291).

- 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 (p. 291).

**Note**

Fargate tasks do not support the DAEMON scheduling strategy.

For more information, see Service Scheduler Concepts (p. 290).

Type: String

Valid values: REPLICA | DAEMON

Default value: REPLICA

Required: No

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| --tags key1=value1,key2=value2 | Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: key1=value1,key2=value2,key3=value3. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the --disable-ecs-managed-tags flag. For more information, see Tagging Resources (p. 528).

Type: Key value pairs

Required: No

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--disable-ecs-managed-tags</td>
<td>Disable the Amazon ECS managed tags. For more information, see Tagging Your Resources for Billing (p. 361).</td>
</tr>
</tbody>
</table>

Required: No
Using a Load Balancer

You can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service. For more information, see Service Load Balancing (p. 306). After you create a service, you can’t change the load balancer name or target group ARN, container name, and container port specified in the service definition.

**Note**

You must create your load balancer resources before you can configure a service to use them. Your load balancer resources should reside in the same VPC as your container instances, and they should be configured to use the same subnets. You must also add a security group rule to your container instance security group that allows inbound traffic from your load balancer. For more information, see Creating a Load Balancer (p. 311).

- To configure your service to use an existing Elastic Load Balancing Classic Load Balancer, you must specify the load balancer name, 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 is registered with the load balancer specified here.

- To configure your service to use an existing Elastic Load Balancing Application Load Balancer, you must specify 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 here.

The `--health-check-grace-period` option specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started. This is valid only if your service is configured to use a load balancer. If your tasks take a while to start and respond to Elastic Load Balancing health checks, you can specify a health check grace period of up to 1,800 seconds during which the Amazon ECS service scheduler ignores the Elastic Load Balancing health check status. This grace period can prevent the Amazon ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

### Using Service Discovery

Your Amazon ECS service can optionally be configured to use Amazon ECS Service Discovery. Service discovery uses Amazon Route 53 auto naming API actions to manage DNS entries for your service’s tasks, making them discoverable within your VPC. For more information, see Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 468).

### Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see AWS Tagging Strategies.

When using the `ecs-cli compose service create` command, using the `--tags` flag allows you to add metadata tags to the task definition and service. The tags are added to the service and task definition when the resources are created. The tags are propagated from your task definition to tasks created by the service. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the `--disable-ecs-managed-tags` flag. For more information, see Tagging Your Resources for Billing (p. 361).

### `ecs-cli compose service start`

Starts one copy of each of the containers on the created Amazon ECS service. This command updates the desired count of the service to 1.

#### Syntax

```
ecs-cli compose service start [--create-log-groups] [--force-deployment] [--help]
```

#### Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--timeout value</code></td>
<td>Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--create-log-groups</td>
<td>Creates the CloudWatch log groups specified in your Compose files.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--force-new-deployment</td>
<td>Forces a new deployment of the service.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster-config</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use.</td>
</tr>
<tr>
<td>cluster_config_name</td>
<td>Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--ecs-profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use.</td>
</tr>
<tr>
<td>ecs_profile</td>
<td>Defaults to the profile configured using the configure profile command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--aws-profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials.</td>
</tr>
<tr>
<td>aws_profile</td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster, -c cluster_name</td>
<td>Specifies the Amazon ECS cluster name to use.</td>
</tr>
<tr>
<td></td>
<td>Defaults to the cluster configured using the configure command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

Default value: 5
ecs-cli compose service up

Creates an Amazon ECS service from your compose file (if it does not already exist) and runs one instance of that task on your cluster (a combination of the create and start commands). This command updates the desired count of the service to 1.

Syntax


Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--deployment-max-percent</td>
<td>Specifies the upper limit (as a percentage of the service's desiredCount) of the number of running tasks that can be running in a service during a deployment. For more information, see maximumPercent (p. 295).</td>
</tr>
<tr>
<td>Default value: 200</td>
<td>Required: No</td>
</tr>
<tr>
<td>--deployment-min-healthy-percent</td>
<td>Specifies the lower limit (as a percentage of the service's desiredCount) of the number of running tasks that must remain running and healthy in a service during a deployment. For more information, see minimumHealthyPercent (p. 296).</td>
</tr>
<tr>
<td>Default value: 100</td>
<td>Required: No</td>
</tr>
<tr>
<td>--load-balancer-name value</td>
<td>--target-group-arn value</td>
</tr>
<tr>
<td>Required: No</td>
<td></td>
</tr>
<tr>
<td>--container-name</td>
<td>Specifies the container name (as it appears in a container definition). This parameter is required if a load balancer or target group is specified.</td>
</tr>
<tr>
<td>Required: No, unless a load balancer or target group is specified.</td>
<td></td>
</tr>
<tr>
<td>--container-port</td>
<td>Specifies the port on the container to associate with the load balancer. This port must correspond</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
| | to a `containerPort` in the service's task definition. This parameter is required if a load balancer or target group is specified. Required: No, unless a load balancer or target group is specified.
| `--load-balancer-name` | Specifies the name of a previously configured Elastic Load Balancing load balancer to associate with your service. Required: No
| `--role` | Specifies the name or full Amazon Resource Name (ARN) of the IAM role that allows Amazon ECS to make calls to your load balancer or target group on your behalf. This parameter is required if you're using a load balancer or target group with your service. If you specify the role parameter, you must also specify a load balancer name or target group ARN, along with a container name and container port. Required: No, unless a load balancer or target group is specified.
| `--health-check-grace-period` | Specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started. Required: No
| `--create-log-groups` | Creates the CloudWatch log groups specified in your Compose files. Required: No
| `--enable-service-discovery` | Specifies whether to enable service discovery for this service. Required: No
| `--vpc` | Specifies the VPC that will be attached to the private DNS namespace for service discovery. This parameter is required if `--private-dns-namespace` is specified. Required: No
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--private-dns-namespace</td>
<td>Specifies the name of the private DNS namespace to use with service discovery. The Amazon ECS CLI automatically creates the namespace if it doesn't exist. For example, if the namespace is corp, a service named foo is reachable via DNS at foo.corp. If you use this parameter, you must also specify a VPC using the --vpc parameter. Required: No</td>
</tr>
<tr>
<td>--private-dns-namespace-id</td>
<td>Specifies the ID of an existing private DNS namespace to use with service discovery. If you use this parameter, you can't specify either --private-dns-namespace or --vpc. Required: No</td>
</tr>
<tr>
<td>--public-dns-namespace</td>
<td>Specifies the name of the public DNS namespace to use with service discovery. For example, if the namespace is corp, a service named foo is reachable via DNS at foo.corp.                                    Required: No</td>
</tr>
<tr>
<td>--public-dns-namespace-id</td>
<td>Specifies the ID of an existing public DNS namespace to use with service discovery. If you use this parameter, you can't specify a --public-dns-namespace. Required: No</td>
</tr>
<tr>
<td>--sd-container-name</td>
<td>Specifies the name of the container, which is referred to as a service in your Docker Compose file. For more information, see Service configuration reference. This parameter is required if you're using SRV records. Required: No, unless SRV DNS records are being used.</td>
</tr>
<tr>
<td>--sd-container-port</td>
<td>Specifies the port on the container that will be used for service discovery. This parameter is required if you're using SRV records. Required: No, unless SRV DNS records are being used.</td>
</tr>
<tr>
<td>--dns-ttl</td>
<td>Specifies the amount of time, in seconds, that you want DNS resolvers to cache the settings for the DNS records used for service discovery. Default value: 60. Required: No</td>
</tr>
</tbody>
</table>

API Version 2014-11-13
524
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--dns-type</code></td>
<td>Specifies the type of DNS record used for service discovery. Accepted values are A or SRV. If your task uses either the bridge or host network modes, SRV records are required. If your task uses the awsvpc network mode, A records are the default.</td>
</tr>
<tr>
<td>Required: No</td>
<td></td>
</tr>
<tr>
<td><code>--healthcheck-custom-config-failure-threshold</code></td>
<td>Specifies the number of 30-second intervals that you want the service discovery service to wait after receiving an UpdateInstanceCustomHealthStatus request before it changes the health status.</td>
</tr>
<tr>
<td>Default value: 1</td>
<td></td>
</tr>
<tr>
<td>Required: No</td>
<td></td>
</tr>
<tr>
<td><code>--update-service-discovery</code></td>
<td>If specified, this enables the service discovery service settings for <code>--dns-ttl</code> and <code>--healthcheck-custom-config-failure-threshold</code> to be updated.</td>
</tr>
<tr>
<td>Required: No</td>
<td></td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| \--scheduling-strategy value | Specifies the scheduling strategy to use for the service. 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 (p. 291).  

- 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 (p. 291).  

**Note**  
Fargate tasks do not support the DAEMON scheduling strategy.  
For more information, see Service Scheduler Concepts (p. 290).  
Type: String  
Valid values: REPLICA | DAEMON  
Default value: REPLICA  
Required: No |
| \--tags key1=value1,key2=value2 | Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: key1=value1,key2=value2,key3=value3. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the --disable-ecs-managed-tags flag. For more information, see Tagging Resources (p. 528).  
Type: Key value pairs  
Required: No |
Name | Description
--- | ---
--disable-ecs-managed-tags | Disable the Amazon ECS managed tags. For more information, see Tagging Your Resources for Billing (p. 361).
Required: No
--region, -r region | Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.
Type: String
Required: No
--cluster-config cluster_config_name | Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.
Type: String
Required: No
--ecs-profile ecs_profile | Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.
Type: String
Required: No
--aws-profile aws_profile | Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials.
Type: String
Required: No
--cluster, -c cluster_name | Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.
Type: String
Required: No
--help, -h | Shows the help text for the specified command.
Required: No

Using a Load Balancer

You can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service. For more information, see Service Load Balancing (p. 306). After you create a service, you can’t change the load balancer name or target group ARN, container name, and container port specified in the service definition.
**Note**
You must create your load balancer resources before you can configure a service to use them. Your load balancer resources should reside in the same VPC as your container instances, and they should be configured to use the same subnets. You must also add a security group rule to your container instance security group that allows inbound traffic from your load balancer. For more information, see Creating a Load Balancer (p. 311).

- To configure your service to use an existing Elastic Load Balancing Classic Load Balancer, you must specify the load balancer name, 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 is registered with the load balancer specified here.
- To configure your service to use an existing Elastic Load Balancing Application Load Balancer, you must specify 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 here.

The `--health-check-grace-period` option specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started. This is valid only if your service is configured to use a load balancer. If your tasks take a while to start and respond to Elastic Load Balancing health checks, you can specify a health check grace period of up to 1,800 seconds during which the Amazon ECS service scheduler ignores the Elastic Load Balancing health check status. This grace period can prevent the Amazon ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

**Using Service Discovery**

Your Amazon ECS service can optionally be configured to use Amazon ECS Service Discovery. Service discovery uses Amazon Route 53 auto naming API actions to manage DNS entries for your service’s tasks, making them discoverable within your VPC. For more information, see Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 468).

**Tagging Resources**

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see AWS Tagging Strategies.

When using the `ecs-cli compose service up` command, using the `--tags` flag allows you to add metadata tags to the task definition and service. The tags will be added to the service and task definition when the resources are created. The tags will be propagated from your task definition to tasks created by the service. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the `--disable-ecs-managed-tags` flag. For more information, see Tagging Your Resources for Billing (p. 361).

**Examples**

**Example 1**

This example brings up an Amazon ECS service with the project name `hello-world` from the `hello-world.yml` compose file.

```
esc-cli compose --project-name hello-world --file hello-world.yml service up
```

Output:
INFO[0000] Using ECS task definition
TaskDefinition="ecscompose-hello-world:7"
INFO[0000] Created an ECS service
service=ecscompose-service-hello-world
taskDefinition="ecscompose-hello-world:7"
serviceName=ecscompose-service-hello-world
INFO[0015] (service ecscompose-service-hello-world) has started 1 tasks: (task 682dc22f-8bfa-4c28-b6f8-3a916d88f86a). timestamp=2017-08-18 21:16:00 +0000 UTC
INFO[0060] Service status
desiredCount=1
serviceName=ecscompose-service-hello-world
INFO[0060] ECS Service has reached a stable state
desiredCount=1
runningCount=1
serviceName=ecscompose-service-hello-world

Example 2
This example creates a service from the nginx-compose.yml compose file and configures it to use an existing Application Load Balancer.

```bash
csi-cli compose -f nginx-compose.yml service up --target-group-arn arn:aws:elasticloadbalancing:us-east-1:aws_account_id:targetgroup/ecs-cli-alb/9856106fcco54d68 --container-name nginx --container-port 80 --role ecsServiceRole
```

Example 3
This example creates a service from the nginx-compose.yml compose file and configures it to use an existing Application Load Balancer with a health check grace period of 25 seconds.

```bash
csi-cli compose -f nginx-compose.yml service up --target-group-arn arn:aws:elasticloadbalancing:us-east-1:aws_account_id:targetgroup/ecs-cli-alb/9856106fcco54d68 --container-name nginx --container-port 80 --role ecsServiceRole --health-check-grace-period 25
```

ecs-cli compose service ps, list
Lists all the containers in your cluster that belong to the service created with the compose project.

**Syntax**

`ecs-cli compose service ps|list [--desired-status status] [--help]`

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--desired-status <code>status</code></td>
<td>The container desired status to filter the container list results with.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td></td>
<td>Valid values: RUNNING</td>
</tr>
<tr>
<td>--region, <code>-r</code> <code>region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <strong>configure</strong> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

API Version 2014-11-13
529
### Name
- **--cluster-config** *cluster_config_name*
  - Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.
  - Type: String
  - Required: No

- **--ecs-profile** *ecs_profile*
  - Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the `configure profile` command.
  - Type: String
  - Required: No

- **--aws-profile** *aws_profile*
  - Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in `~/.aws/credentials`.
  - Type: String
  - Required: No

- **--cluster**, **-c** *cluster_name*
  - Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the `configure` command.
  - Type: String
  - Required: No

- **--help**, **-h**
  - Shows the help text for the specified command.
  - Required: No

---

### ecs-cli compose service scale

Scales the desired count of the service to the specified count.

**Syntax**

```
ecs-cli compose service scale [--deployment-max-percent n] [--deployment-min-healthy-percent n] [--timeout value] n [--help]
```

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--deployment-max-percent</strong></td>
<td>Specifies the upper limit (as a percentage of the service's <code>desiredCount</code>) of the number of running tasks that can be running in a service during a deployment. For more information, see <code>maximumPercent</code> (p. 295). Default value: 200</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>--deployment-min-healthy-percent</td>
<td>Specifies the lower limit (as a percentage of the service's desiredCount) of the number of running tasks that must remain running and healthy in a service during a deployment. For more information, see <code>minimumHealthyPercent</code> (p. 296). Default value: 100</td>
</tr>
<tr>
<td>--timeout value</td>
<td>Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes). Default value: 5</td>
</tr>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command. Type: String Required: No</td>
</tr>
<tr>
<td>--cluster-config cluster_config_name</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No</td>
</tr>
<tr>
<td>--ecs-profile ecs_profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command. Type: String Required: No</td>
</tr>
<tr>
<td>--aws-profile aws_profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>. Type: String Required: No</td>
</tr>
</tbody>
</table>
### Examples

#### Example 1

This example scales the service created by the `hello-world` project to a desired count of 2.

```
ecs-cli compose --project-name hello-world --file hello-world.yml service scale 2
```

**Output:**

```
INFO[0000] Updated ECS service successfully
desiredCount=2
serviceName=ecscompose-service-hello-world

INFO[0000] Service status
desiredCount=2 runningCount=1
serviceName=ecscompose-service-hello-world

INFO[0030] (service ecscompose-service-hello-world) has started 1 tasks: (task 80602da8-442c-48ea-a8a9-8032b8c32b89). timestamp=2017-08-18 21:17:44 +0000 UTC
desiredCount=2 runningCount=2

INFO[0075] ECS Service has reached a stable state
desiredCount=2 runningCount=2
serviceName=ecscompose-service-hello-world
```

### ecs-cli compose service stop

Stops the running tasks that belong to the service created with the compose project. This command updates the desired count of the service to 0.

The `--timeout` option specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes).

#### Syntax

```
ecs-cli compose service stop [--timeout value] [--help]
```

#### Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--timeout value</code></td>
<td>Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to</td>
</tr>
</tbody>
</table>
## Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change</td>
<td>If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes). Default value: 5 Required: No</td>
</tr>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster-config</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--ecs-profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--aws-profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster, -c cluster_name</td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

### ecs-cli compose service rm, delete, down

Updates the desired count of the service to 0 and then deletes the service.
Syntax

This command accepts rm, delete, or down when used.

```
escs-cli compose service rm|delete|down [--timeout value] [--delete-namespace] [--help]
```

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--timeout value</code></td>
<td>Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes).&lt;br&gt;Default value: 5&lt;br&gt;Required: No</td>
</tr>
<tr>
<td><code>--delete-namespace</code></td>
<td>If specified, the private namespace created with either the <code>compose service create</code> or <code>compose service up</code> commands is deleted.&lt;br&gt;Required: No</td>
</tr>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.&lt;br&gt;Type: String&lt;br&gt;Required: No</td>
</tr>
<tr>
<td><code>--cluster-config cluster_config_name</code></td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.&lt;br&gt;Type: String&lt;br&gt;Required: No</td>
</tr>
<tr>
<td><code>--ecs-profile ecs_profile</code></td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.&lt;br&gt;Type: String&lt;br&gt;Required: No</td>
</tr>
</tbody>
</table>
| `--aws-profile aws_profile`   | Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in `~/.aws/credentials`.  

API Version 2014-11-13

534
ecs-cli logs

Retrieves container logs from CloudWatch Logs. Only valid for tasks that use the awslogs driver and have a log stream prefix specified.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see Installing the Amazon ECS CLI (p. 450).

**Syntax**

```
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cluster, -c cluster_name</td>
<td>Specifies the Amazon ECS cluster name to use.</td>
</tr>
<tr>
<td></td>
<td>Defaults to the cluster configured using the configure command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
</tbody>
</table>

**Examples**

**Example 1**

This example scales the service created by the hello-world project to a desired count of 0 and then deletes the service.

```
css-cli compose --project-name hello-world --file hello-world.yml service rm
```

Output:

```
INFO[0000] Updated ECS service successfully                      desiredCount=0
serviceName=ecscompose-service-hello-world
INFO[0000] Service status                                     desiredCount=0 runningCount=2
serviceName=ecscompose-service-hello-world
INFO[0015] Service status                                     desiredCount=0 runningCount=0
serviceName=ecscompose-service-hello-world
INFO[0015] (service ecscompose-service-hello-world) has stopped 2 running tasks: (task 682dc22f-8bfa-4c28-b6f8-3a916bd8f86a) (task 80602da8-442c-48ea-a8a9-80328c302b89).
timestamp=2017-08-18 21:25:28 +0000 UTC
INFO[0015] ECS Service has reached a stable state              desiredCount=0 runningCount=0
serviceName=ecscompose-service-hello-world
INFO[0015] Deleted ECS service                                service=ecscompose-service-hello-world
INFO[0015] ECS Service has reached a stable state              desiredCount=0 runningCount=0
serviceName=ecscompose-service-hello-world
```
## Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--task-id task_id</code></td>
<td>Prints the logs for this ECS task. Type: String Required: Yes</td>
</tr>
<tr>
<td><code>--task-def task_definition</code></td>
<td>Specifies the name or full Amazon Resource Name (ARN) of the ECS task definition associated with the task ID. This is needed only if the task has been stopped. Type: String Required: No</td>
</tr>
<tr>
<td><code>--follow</code></td>
<td>Specifies if the logs should be streamed. Required: No</td>
</tr>
<tr>
<td><code>--filter-pattern search_string</code></td>
<td>Specifies the substring to search for within the logs. Type: String Required: No</td>
</tr>
<tr>
<td><code>--since n</code></td>
<td>Returns logs newer than a relative duration in minutes. Can't be used with <code>--start-time</code>. Type: Integer Required: No</td>
</tr>
<tr>
<td><code>--start-time timestamp</code></td>
<td>Returns logs after a specific date (format: RFC 3339. Example: <code>2006-01-02T15:04:05+07:00</code>). Can't be used with <code>--since</code> flag. Required: No</td>
</tr>
<tr>
<td><code>--end-time timestamp</code></td>
<td>Returns logs before a specific date (format: RFC 3339. Example: <code>2006-01-02T15:04:05+07:00</code>). Cannot be used with <code>--follow</code>. Required: No</td>
</tr>
<tr>
<td><code>--timestamps</code></td>
<td>Specifies if timestamps are shown on each line in the log output. Required: No</td>
</tr>
<tr>
<td><code>--region, -r region</code></td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command. Type: String Required: No</td>
</tr>
</tbody>
</table>
Table: Parameter Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cluster-config</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No</td>
</tr>
<tr>
<td>--ecs-profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No</td>
</tr>
<tr>
<td>--aws-profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in ~/.aws/credentials. Type: String Required: No</td>
</tr>
<tr>
<td>--cluster, -c cluster_name</td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command. Required: No</td>
</tr>
</tbody>
</table>

**Examples**

**Example**

This example prints the log for a task.

```
ecs-cli logs --task-id task_id
```

The contents of the log is in the output if successful.

**ecs-cli check-attributes**

Checks if a given list of container instances can run a given task definition by checking their attributes. Outputs attributes that are required by the task definition but not present on the container instances.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI](p. 450).
Syntax

```
ecs-cli check-attributes [--task-def task_definition] [--container-instances value] [--help]
```

Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--task-def task_definition</td>
<td>Specifies the name or full Amazon Resource Name (ARN) of the ECS task definition associated with the task ID. This is only needed if the task has been stopped.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--container-instances value</td>
<td>A list of container instance IDs or full ARN entries to check if all required attributes are available for the Task Definition to RunTask.</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster-config</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td>cluster_config_name</td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--ecs-profile ecs_profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--aws-profile aws_profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster, -c cluster_name</td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
</tbody>
</table>
### Examples

#### Example

This example checks multiple container instances and verifies that they contain the attributes necessary to successfully run the specified task definition.

```
ecs-cli check-attributes --container-instances 28c5abd2-360e-41a0-81d8-0afca2d08d9b,45510138-f24f-47c6-a418-71c46dd51f88 --cluster default --region us-east-2 --task-def fluentd-test
```

**Output:**

<table>
<thead>
<tr>
<th>Container Instance</th>
<th>Missing Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>28c5abd2-360e-41a0-81d8-0afca2d08d9b</td>
<td>com.amazonaws.ecs.capability.logging-driver.fluentd</td>
</tr>
<tr>
<td>45510138-f24f-47c6-a418-71c46dd51f88</td>
<td>None</td>
</tr>
</tbody>
</table>

### ecs-cli registry-creds

Facilitates the creation and use of private registry credentials within Amazon ECS. For more information, see [Private Registry Authentication for Tasks](#).

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI](#).

#### Syntax

```
```

#### Options

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--region, -r region</td>
<td>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td></td>
<td>Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: No</td>
</tr>
<tr>
<td>--cluster-config</td>
<td>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</td>
</tr>
<tr>
<td>cluster_config_name</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--ecs-profile</td>
<td>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.</td>
</tr>
<tr>
<td>-c, --cluster</td>
<td>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Shows the help text for the specified command.</td>
</tr>
<tr>
<td>--aws-profile</td>
<td>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</td>
</tr>
</tbody>
</table>

### Available Subcommands

The `ecs-cli registry-creds` command supports the following subcommands. Each of these subcommands has their own flags associated with them, which can be displayed using the `--help` flag.

#### up

Generates AWS Secrets Manager secrets and an IAM task execution role for use in an Amazon ECS task definition. For more information, see `ecs-cli registry-creds up (p. 540).`

#### help

Shows the help text for the specified command.

### ecs-cli registry-creds up

Generates AWS Secrets Manager secrets and an IAM task execution role for use in an Amazon ECS task definition.

**Important**

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI (p. 450)].
**Syntax**

`ecs-cli registry-creds up ./creds_input_file.yml --role-name value [--update-existing-secrets] [--no-role] [--no-output-value] [--output-dir value] [--tags key1=value1,key2=value2] [--help]`

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>./creds_input_file.yml</code></td>
<td>Specifies the values related to private registry authentication. For more information, see Using Private Registry Authentication (p. 542). Required: Yes</td>
</tr>
<tr>
<td><code>--role-name value</code></td>
<td>The name to use for the new task execution role. If the role already exists, new policies are attached to the existing role. For more information, see Amazon ECS Task Execution IAM Role (p. 418). Note: We recommend creating a new task execution role specific to each application to avoid granting permissions to your secrets for applications that do not need them. Required: Yes</td>
</tr>
<tr>
<td><code>--update-existing-secrets</code></td>
<td>Specifies whether existing secrets should be updated with new credential values. Required: No</td>
</tr>
<tr>
<td><code>--no-role</code></td>
<td>If specified, no task execution role is created. Required: No</td>
</tr>
<tr>
<td><code>--no-output-file</code></td>
<td>If specified, no output file for use with <code>compose</code> is created. Required: No</td>
</tr>
<tr>
<td><code>--output-dir value</code></td>
<td>The directory where the output file should be created. If none specified, the file is created in the current working directory. Required: No</td>
</tr>
<tr>
<td><code>--tags key1=value1,key2=value2</code></td>
<td>Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: <code>key1=value1,key2=value2,key3=value3</code>. For more information, see Tagging Resources (p. 543). Type: Key value pairs Required: No</td>
</tr>
<tr>
<td><code>--help, -h</code></td>
<td>Shows the help text for the specified command. Required: No</td>
</tr>
</tbody>
</table>
Using Private Registry Authentication

When using the `ecs-cli registry-creds` up command to manage your private registry authentication credentials, there are certain fields that are specified using an input file. You must specify a file name or path to an input file when using this command.

Currently, the file supports the following schema:

```
version: 1
registry_credentials:
  registry_name:
    secrets_manager_arn: string
    username: string
    password: string
    kms_key_id: string
    container_names:
      - string
```

The following are descriptions for each of these fields.

**registry_name**

Used as the secret name when creating a new secret or updating an existing secret. The secret name must be ASCII letters, digits, or any of the following characters: /_+=.@-. The Amazon ECS CLI adds a prefix to the secret name to indicate that it was created by the CLI. For more information, see `CreateSecret`.

Required: No

**secrets_manager_arn**

The full ARN of an existing secret. Used to specify or update an existing secret. Must be in the following format:

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret_name
```

Required: No

**username**

Specifies the user name for the private registry. We recommend using environment variables for the user name to ensure that no sensitive information is stored in the input file. When using environment variables, use the format `${VAR_NAME}`.

Required: No

**password**

Specifies the password for the private registry. We recommend using environment variables for the password to ensure that no sensitive information is stored in the input file. When using environment variables, use the format `${VAR_NAME}`.

Required: No

**kms_key_id**

Specifies the ARN, Key ID, or alias of the AWS KMS customer master key (CMK) to be used to encrypt the secret. For more information, see `CreateSecret`.

Required: No

**container_names**

Corresponds to a service name in a Docker compose file. For more information, see `ecs-cli compose` (p. 500) or `ecs-cli compose service` (p. 511).
Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see AWS Tagging Strategies.

When using the `ecs-cli registry-creds up` command, using the `--tags` flag enables you to add metadata tags to the Secrets Manager secrets and then IAM roles.

**Note**
Existing Secrets Manager secrets within your account will be tagged, but IAM roles can only be tagged during creation. If you're using an existing IAM role, new tags can't be added.

Examples

Create a Secret with Private Registry Authentication Credentials

This example creates a secret with the private registry credentials specified in the `creds_input.yml` input file.

Create a private registry credentials file, named `creds_input.yml` that contains the user name and password for the private registry as well as the name of the container that will use the private registry credentials. We recommend using environment variables for the credentials to ensure that no sensitive information is stored in the input file. The container name in this file corresponds to the service name in the Docker compose file.

```yaml
version: '1'
registry_credentials:
  dockerhub:
    username: ${MY_REPO_USERNAME}
    password: ${MY_REPO_PASSWORD}
    container_names:
      - database
```

**Important**
We recommend using environment variables for the password to ensure that no sensitive information is stored in the input file. If your input file contains sensitive information, make sure that you delete it after use.

Create the secret. This command creates a secret using the name from the input file, in this example it is `dockerhub`. The Amazon ECS CLI adds a prefix to the secret name to indicate that it was created by the CLI. You also specify the name of your task execution role.

```
ecs-cli registry-creds up ./creds_input.yml --role-name secretsTaskExecutionRole
```

Output:

```
INFO[0000] Processing credentials for registry dockerhub...
INFO[0000] Creating resources for task execution role ecsTaskExecutionRole...
INFO[0000] Created new task execution role arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole
INFO[0000] Attached AWS managed policy arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy to role ecsTaskExecutionRole
```
An output file is created by this command that contains the task execution role name, the ARN of the secret that was created, and the container name. This file is specified using the `--registry-creds` option when using either the `ecs-cli compose` or `ecs-cli compose service` commands. For more information, see `ecs-cli compose` (p. 500) or `ecs-cli compose service` (p. 511).

The following is an example output file:

```yaml
version: "1"
registry_credential_outputs:
  task_execution_role: secretsTaskExecutionRole
  container_credentials:
    dockerhub:
      container_names:
        - database
```

### Create a Secret with Private Registry Authentication Credentials That Use a KMS Key

This example creates a secret with the private registry credentials that are encrypted using a KMS key specified in the `creds_input.yml` input file.

Create a private registry credentials file, named `creds_input.yml` that contains the user name and password for the private registry as well as the name of the container that will use the private registry credentials. We recommend using environment variables for the credentials to ensure that no sensitive information is stored in the input file. The specified KMS key ARN encrypts the values when storing the secret. The container name in this file corresponds to the service name `database` in the Docker compose file.

```yaml
version: '1'
registry_credentials:
  dockerhub:
    username: ${MY_REPO_USERNAME}
    password: ${MY_REPO_PASSWORD}
    kms_key_id: kmsKeyARN
    container_names:
      - database
```

**Important**

We recommend using environment variables for the password to ensure that no sensitive information is stored in the input file. If your input file contains sensitive information, make sure that you delete it after use.

### Create Multiple Secrets For Multiple Private Registries

This example creates multiple secrets with the private registry credentials for multiple registries.

Create a private registry credentials file, named `creds_input.yml` that contains the credentials from two different private registries. Each set of credentials are used to create its own secret. This example also shows two different containers using one secret.

```yaml
version: '1'
registry_credentials:
  dockerhub:
```

API Version 2014-11-13

544
username: ${MY_REPO_USERNAME}
password: ${MY_REPO_PASSWORD}
container_names:
  - prod
  - dev
quay.io:
username: ${MY_REPO_USERNAME}
password: ${MY_REPO_PASSWORD}
container_names:
  - database

Important
We recommend using environment variables for the password to ensure that no sensitive
information is stored in the input file. If your input file contains sensitive information, make sure
that you delete it after use.

Using Docker Compose File Syntax

The ecs-cli compose and ecs-cli compose service commands allow you to create task
definitions and manage your Amazon ECS tasks and services using Docker compose files. For more
information, see ecs-cli compose (p. 500) and ecs-cli compose service (p. 511).

At this time, the latest version of the Amazon ECS CLI supports Docker compose file syntax versions 1, 2,
and 3. Only major versions of the Docker compose file syntax are supported. Consequently, the version
specified in the compose file must be the string "1", "1.0", "2", "2.0", "3", or "3.0".

By default, the Amazon ECS CLI commands look for a Docker compose file in the current directory,
named docker-compose.yml. However, you can also specify a different file name or path to a compose
file with the --file option. This is especially useful for managing tasks and services from multiple
compose files at a time with the Amazon ECS CLI.

The following parameters are supported in Compose files for the Amazon ECS CLI:

- cap_add (not valid for tasks using the Fargate launch type)
- cap_drop (not valid for tasks using the Fargate launch type)
- command
- cpu_shares
  
  Note
  If you're using the Compose version 3 format, cpu_shares should be specified in the ecs-
  params.yml file. For more information, see Using Amazon ECS Parameters (p. 546).

- devices (not valid for tasks using the Fargate launch type)
- dns
- dns_search
- entrypoint
- environment: If an environment variable value isn't specified in the Compose file, but it exists in the
  shell environment, the shell environment variable value is passed to the task definition that is created
  for any associated tasks or services.

  Important
  We don't recommend using plaintext environment variables for sensitive information, such as
  credential data.

- env_file

  Important
  We don't recommend using plaintext environment variables for sensitive information, such as
  credential data.
• extends (Compose file version 1 and 2 only)
• extra_hosts
• healthcheck (Compose file version 3 only)

Note
The start_period field isn't supported using the Compose file. To specify a start_period, use the ecs-params.yml file. For more information, see Using Amazon ECS Parameters (p. 546).

• hostname
• image
• labels
• links (not valid for tasks using the Fargate launch type)
• log_driver (Compose file version 1 only)
• log_opt (Compose file version 1 only)
• logging (Compose file version 2 and 3)
  • driver
  • options
• mem_limit (in bytes)

Note
If you’re using the Compose version 3 format, mem_limit should be specified in the ecs-params.yml file. For more information, see Using Amazon ECS Parameters (p. 546).

• mem_reservation (in bytes)

Note
If you’re using the Compose version 3 format, mem_reservation should be specified in the ecs-params.yml file. For more information, see Using Amazon ECS Parameters (p. 546).

• ports
• privileged (not valid for tasks using the Fargate launch type)
• read_only
• security_opt
• shm_size (Compose file version 1 and 2 only and not valid for tasks using the Fargate launch type)
• tmpfs (not valid for tasks using the Fargate launch type)
• tty
• ulimits
• user
• volumes
• volumes_from (Compose file version 1 and 2 only)
• working_dir

Important
The build directive isn’t supported at this time.

For more information about Docker Compose file syntax, see the Compose file reference in the Docker documentation.

Using Amazon ECS Parameters

When using the ecs-cli compose or ecs-cli compose service commands to manage your Amazon ECS tasks and services, there are certain fields in an Amazon ECS task definition that do not correspond to fields in a Docker compose file. You can specify those values using an ECS parameters file
with the --ecs-params flag. By default, the command looks for an ECS parameters file in the current directory named ecs-params.yml. However, you can also specify a different file name or path to an ECS parameters file with the --ecs-params option.

Currently, the file supports the follow schema:

```
version: 1

task_definition:
  ecm_network_mode: string
  task_role_arn: string
  task_execution_role: string
  task_size:
    cpu_limit: string
    mem_limit: string
  pid_mode: string
  ipc_mode: string

services:
  <service_name>:
    essential: boolean
    repository_credentials:
      credentials_parameter: string
    cpu_shares: integer
    mem_limit: string
    mem_reservation: string
    gpu: string
    init_process_enabled: boolean
    healthcheck:
      test: ["CMD", "curl -f http://localhost"]
      interval: string
      timeout: string
      retries: integer
      start_period: string
    secrets:
      - value_from: string
        name: string
    docker_volumes:
      - name: string
        scope: string
        autoprovision:
        driver: string
        driver_opts: boolean
        string: string
    labels:
      string: string

run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - subnet_id1
        - subnet_id2
      security_groups:
        - secgroup_id1
        - secgroup_id2
      assign_public_ip: ENABLED

task_placement:
  strategy:
    - type: string
      field: string
  constraints:
    - type: string
      expression: string
  service_discovery:
    container_name: string
    container_port: integer
    private_dns_namespace:
```

API Version 2014-11-13

547
vpc: string
id: string
name: string
description: string
public_dns_namespace:
  id: string
  name: string
service_discovery_service:
  name: string
description: string
dns_config:
  type: string
ttl: integer
healthcheck_custom_config:
  failure_threshold: integer

The fields listed under task_definition correspond to fields to be included in your Amazon ECS task definition.

- **ecs_network_mode** – Corresponds to networkMode in an ECS task definition. Supported values are none, bridge, host, or awsvpc. The default value is bridge. If you are using task networking, this field must be set to awsvpc. For more information, see Network Mode (p. 196).
- **task_role_arn** – The name or full ARN of an IAM role to be associated with the task. For more information, see Task Role (p. 196).
- **task_execution_role** – The name or full ARN of the task execution role. This is a required field if you want your tasks to be able to store container application logs in CloudWatch or allow your tasks to pull container images from Amazon ECR. For more information, see Amazon ECS Task Execution IAM Role (p. 418).
- **task_size** – The CPU and memory values for the task. If you are using the EC2 launch type, this field is optional and any value can be used. If using the Fargate launch type, this field is required and you must use one of the following sets of values for the cpu and memory parameters.

<table>
<thead>
<tr>
<th>CPU value</th>
<th>Memory value (MiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 (.25 vCPU)</td>
<td>512 (0.5GB), 1024 (1GB), 2048 (2GB)</td>
</tr>
<tr>
<td>512 (.5 vCPU)</td>
<td>1024 (1GB), 2048 (2GB), 3072 (3GB), 4096 (4GB)</td>
</tr>
<tr>
<td>1024 (1 vCPU)</td>
<td>2048 (2GB), 3072 (3GB), 4096 (4GB), 5120 (5GB), 6144 (6GB), 7168 (7GB), 8192 (8GB)</td>
</tr>
<tr>
<td>2048 (2 vCPU)</td>
<td>Between 4096 (4GB) and 16384 (16GB) in increments of 1024 (1GB)</td>
</tr>
<tr>
<td>4096 (4 vCPU)</td>
<td>Between 8192 (8GB) and 30720 (30GB) in increments of 1024 (1GB)</td>
</tr>
</tbody>
</table>

For more information, see Task Size (p. 223).
- **pid_mode** – The process namespace to use for the containers in the task. The valid values are host or task. If host is specified, then all containers within the tasks that specified the host PID mode on the same container instance share the same IPC resources 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 more information, see PID settings in the Docker run reference.

If the host PID mode is used, be aware that there is a heightened risk of undesired process namespace expose. For more information, see Docker security.
Note
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

- **ipc_mode** – 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 containers 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 containers 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. For more information, see `IPC settings` in the Docker run reference.

If the `host` IPC mode is used, be aware that there is a heightened risk of undesired IPC namespace expose. For more information, see Docker security.

If you are setting namespaced kernel parameters using `systemControls` for the containers in the task, the following will apply to your IPC resource namespace. For more information, see System Controls in the Amazon Elastic Container Service Developer Guide.

- For tasks that use the `host` IPC mode, IPC namespace related `systemControls` are not supported.
- For tasks that use the `task` IPC mode, IPC namespace related `systemControls` will apply to all containers within a task.

Note
This parameter is not supported for Windows containers or tasks using the Fargate launch type.

- **services** – Corresponds to the services listed in your Docker compose file, with `service_name` matching the name of the container to run. Its fields are merged into a container definition.

- **essential** – If the `essential` parameter of a container is marked as `true`, and that container fails or stops for any reason, 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 does not affect the rest of the containers in a task. The default value is `true`.

All tasks must have at least one essential container. If you have an application that is composed of multiple containers, you should group containers that are used for a common purpose into components, and separate the different components into multiple task definitions.

- **repository_credentials** – If you are using a private repository for pulling images, `repository_credentials` allows you to specify an AWS Secrets Manager secret ARN for the name of the secret containing your private repository credentials as a `credential_parameter`. For more information, see Private Registry Authentication for Tasks (p. 251).

- **cpu_shares** – This parameter maps to `cpu_shares` in the Docker compose file reference. If you are using Docker compose version 3, this field is optional and must be specified in the ECS params file rather than the compose file. In Docker compose version 2, this field can be specified in either the compose or ECS params file. If it is specified in the ECS params file, the value overrides the value present in the compose file.

- **mem_limit** – This parameter maps to `mem_limit` in the Docker compose file reference. If you are using Docker compose version 3, this field is optional and must be specified in the ECS params file rather than the compose file. In Docker compose version 2, this field can be specified in either the compose or ECS params file. If it is specified in the ECS params file, the value overrides the value present in the compose file.

- **mem_reservation** – This parameter maps to `mem_reservation` in the Docker compose file reference. If you are using Docker compose version 3, this field is optional and must be specified in the ECS params file rather than the compose file. In Docker compose version 2, this field can be specified in either the compose or ECS params file. If it is specified in the ECS params file, the value overrides the value present in the compose file.
• **gpu** – The number of physical GPUs the Amazon ECS container agent will reserve for the container. This parameter maps to the `resourceRequirements` field in a task definition. For more information, see [Working with GPUs on Amazon ECS](p. 38).

• **init_process_enabled** – This parameter enables you to 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.

• **healthcheck** – This parameter maps to `healthcheck` in the [Docker compose file reference](https://docs.docker.com/compose/compose-file/). The `test` field can also be specified as `command` and must be either a string or a list. If it’s a list, the first item must be either `NONE`, `CMD`, or `CMD-SHELL`. If it’s a string, it’s equivalent to specifying `CMD-SHELL` followed by that string. The `interval`, `timeout`, and `start_period` fields are specified as durations in a string format. For example: `2.5s`, `10s`, `1m30s`, `2h23m`, or `5h34m56s`.

**Note**
If no units are specified, seconds are assumed. For example, you can specify either `10s` or simply `10`.

• **secrets** – This parameter 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. For more information, see [Specifying Sensitive Data](p. 254).

• **value_from** – This is the AWS Systems Manager Parameter Store ARN or name to expose to the container. 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 secret. If the parameter exists in a different Region, then the full ARN must be specified.

• **name** – The value to set as the environment variable on the container.

• **docker_volumes** – This parameter allows you to create docker volumes. The `name` key is required, and `scope`, `autopropvision`, `driver`, `driver_opts` and `labels` correspond with the Docker volume configuration fields in a task definition. For more information, see [DockerVolumeConfiguration](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/dockervolumeconfiguration.html) in the *Amazon Elastic Container Service API Reference*. Volumes defined with the `docker_volumes` key can be referenced in your compose file by name, even if they were not also specified in the compose file.

The fields listed under `run_params` are for values needed as options to any API calls not specifically related to a task definition, such as `compose up (RunTask)` and `compose service up (CreateService)`. Currently, the only supported parameter under `run_params` is `network_configuration`, which is a required parameter to use task networking and when using tasks with the Fargate launch type.

• **network_configuration** – Required if you specified `awsvpc` for `ecs_network_mode`. It uses one nested parameter, `awsvpc_configuration`, which has the following subfields:

  • **subnets** – A list of subnet IDs used to associate with your tasks. The listed subnets must be in the same VPC and Availability Zone as the instances on which to launch your tasks.

  • **security_groups** – A list of security group IDs to associate with your tasks. The listed security must be in the same VPC as the instances on which to launch your tasks.

  • **assign_public_ip** – The supported values for this field are `ENABLED` or `DISABLED`. This field is only used for tasks using the Fargate launch type. If this field is present in tasks using task networking with the EC2 launch type, the request fails.

• **task_placement** – This parameter allows you to specify task placement options. It is optional if you are using the EC2 launch type. It is not supported Fargate launch type. For more information, see [Amazon ECS Task Placement](p. 274).

It has the following subfields:

• **strategy** – A list of objects, with two keys. Valid keys are `type` and `field`.

  • **type** – Valid values are `random`, `binpack`, or `spread`. If `random` is specified, the `field` key should not be provided.
- **field** – Valid values depend on the strategy type.
  - For **spread**, valid values are `instanceId`, `host`, or attribute key-value pairs, for example `attribute:ecs.instance-type =~ t2.*`.
  - For **binpack**, valid values are `cpu` or `memory`.
- **constraints** – A list of objects, with two keys. Valid keys are `type` and `expression`.
  - **type** – Valid values are `distinctInstance` and `memberOf`. If `distinctInstance` is specified, the `expression` key should not be provided.
  - **expression** – When type is `memberOf`, valid values are key-value pairs for attributes or task groups, for example `task:group == databases` or `attribute:color =~ green`.
- **service_discovery** – This parameter allows you to configure Amazon ECS Service Discovery using Amazon Route 53 auto naming API actions to manage DNS entries for your service's tasks. For more information, see Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 468).
Using the AWS CLI with Amazon ECS

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts. For more information on the AWS CLI, see http://aws.amazon.com/cli/.

For more information on the other tools available for managing your AWS resources, including the different AWS SDKs, IDE toolkits, and the Windows PowerShell command line tools, see http://aws.amazon.com/tools/.

The following steps help you set up an Amazon ECS cluster using either a Fargate or EC2 task:

Topics
- Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI (p. 552)
- Tutorial: Creating a Cluster with an EC2 Task Using the AWS CLI (p. 558)

Tutorial: Creating a Cluster with a Fargate Task Using 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. 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 the AWS Command Line Interface.

Topics
- Prerequisites (p. 552)
- Step 1: (Optional) Create a Cluster (p. 553)
- Step 2: Register a Task Definition (p. 553)
- Step 3: List Task Definitions (p. 555)
- Step 4: Create a Service (p. 555)
- Step 5: List Services (p. 556)
- Step 6: Describe the Running Service (p. 557)

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 the AWS Command Line Interface.
- The steps in Setting Up with Amazon ECS (p. 7) have been completed.
- Your AWS user has the required permissions specified in the Amazon ECS First Run Wizard (p. 440) IAM policy example.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.
Step 1: (Optional) 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:

```bash
aws ecs create-cluster --cluster-name fargate-cluster
```

**Output:**

```json
{
    "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 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. For more information about the available task definition parameters, see Amazon ECS Task Definitions (p. 186).

```json
{
    "family": "sample-fargate",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "fargate-app",
            "image": "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> Amazon ECS Sample App</h1> Congratulations!
```
Step 2: Register a Task Definition

Your application is now running on a container in Amazon ECS.

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 as in the below example. 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/fargate-task.json
```

The `register-task-definition` returns a description of the task definition after it completes its registration.

```
{
  "taskDefinition": {
    "status": "ACTIVE",
    "networkMode": "awsvpc",
    "family": "sample-fargate",
    "placementConstraints": [],
    "requiresAttributes": [
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
      },
      {
        "name": "ecs.capability.task-eni"
      }
    ],
    "cpu": "256",
    "compatibilities": ["EC2", "FARGATE"],
    "volumes": [],
    "memory": "512",
    "requiresCompatibilities": ["FARGATE"],
    "taskDefinitionArn": "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:2",
    "containerDefinitions": [
      {
        "environment": [],
        "name": "fargate-app",
        "mountPoints": [],
        "image": "httpd:2.4",
        "cpu": 0,
        "portMappings": [
          {
            "protocol": "tcp",
```
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`.

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

Output:

```
{
  "taskDefinitionArns": [
    "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:2"
  ]
}
```

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 where at least two instances of the `sample-fargate:1` task definition are kept running in your cluster.

```bash
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --task-definition sample-fargate:1 --desired-count 2 --launch-type "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-abcd1234]}"
```

Output:

```
{
  "service": {
    "status": "ACTIVE",
    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:1"
  }
}
```
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.
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
```

Output:

```
{
  "services": [
    {
      "status": "ACTIVE",
      "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate: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",
      "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
      "deploymentConfiguration": {
        "maximumPercent": 200,
        "minimumHealthyPercent": 100
      },
      "deployments": [
        {
          "status": "PRIMARY",
          "networkConfiguration": {
            "awsvpcConfiguration": {
              "subnets": [
                "subnet-abcd1234"
              ]
            }
          }
        }
      ]
    }
  ]
}
```
Tutorial: Creating a Cluster with an EC2 Task Using 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. 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 the AWS Command Line Interface.

Topics
- Prerequisites (p. 559)
- Step 1: (Optional) Create a Cluster (p. 559)
- Step 2: Launch an Instance with the Amazon ECS AMI (p. 559)
- Step 3: List Container Instances (p. 561)
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 the AWS Command Line Interface.
- The steps in Setting Up with Amazon ECS (p. 7) have been completed.
- Your AWS user has the required permissions specified in the Amazon ECS First Run Wizard (p. 440) IAM policy example.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

Step 1: (Optional) 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"
  }
}
```

Step 2: Launch an Instance with the Amazon ECS AMI

You must have an Amazon ECS container instance in your cluster before you can run tasks on it. If you do not have any container instances in your cluster, see Launching an Amazon ECS Container Instance (p. 101) for more information.

The following table lists the current Amazon ECS-optimized Amazon Linux AMI IDs by Region.
<table>
<thead>
<tr>
<th>Region</th>
<th>AMI Name</th>
<th>AMI ID</th>
<th>EC2 Console Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-06a8ae0ecd30e804</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0750ab1027b6314c</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-03fe84be94ca9cc1</td>
<td>Launch instance</td>
</tr>
<tr>
<td>us-west-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-043c4e6b6752b99</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0d772c70a2d689e8</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-062ef2a2561c9364</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005600074f3ada4</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0a80c5ae873c08c64</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0b78ef7facf3f93a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0c5058003c511da15</td>
<td>Launch instance</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005ff694f167b58</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0005ce40ccf58b98</td>
<td>Launch instance</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0066a513748afa1e</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-06a20f16dd2f5074</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-087de2c1b54c6bd9</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-09b156894255325f</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-03ca259ae4cb8683</td>
<td>Launch instance</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-055c29a7d5fc2d48a</td>
<td>Launch instance</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>amzn-ami-2018.03.q-amazon-ecs-optimized</td>
<td>ami-0d83f147ba8afa3c</td>
<td>Launch instance</td>
</tr>
</tbody>
</table>
Within a few minutes of launching your container instance, the Amazon ECS agent registers the instance with your default cluster. You can list the container instances in a cluster by running the following command:

```
aws ecs list-container-instances --cluster default
```

Output:

```
{
  "containerInstanceArns": [
    "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID"
  ]
}
```

### Step 4: 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 default --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",
          "type": "INTEGER",
          "doubleValue": 0.0
        }
      ]
    }
  ]
}
```
Step 4: Describe your Container Instance

```

"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,
"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"
  }
]
```

API Version 2014-11-13
562
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.

**Step 5: Register a 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 uses a `busybox` image from Docker Hub and simply sleeps for 360 seconds. For more information about the available task definition parameters,

```json
{
  "containerDefinitions": [
    {
      "name": "sleep",
      "image": "busybox",
      "cpu": 10,
      "command": ["sleep", "360"],
      "memory": 10,
      "essential": true
    }
  ],
  "family": "sleep360"
}
```

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 as in the below example. 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/sleep360.json

To use a JSON string for container definitions:

aws ecs register-task-definition --family sleep360 --container-definitions "[{
  "name": "sleep",
  "image": "busybox",
  "cpu": 10,
  "command": ["sleep", "360"],
  "memory": 10,
  "essential": true
}]

The `register-task-definition` returns a description of the task definition after it completes its registration.

```
{
  "taskDefinition": {
    "volumes": [],
    "taskDefinitionArn": "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep360:1",
    "containerDefinitions": [
      {
        "environment": [],
        "name": "sleep",
        "mountPoints": [],
        "image": "busybox",
        "cpu": 10,
        "portMappings": [],
        "command": ["sleep", "360"],
        "memory": 10,
        "essential": true,
        "volumesFrom": []
      }
    ],
    "family": "sleep360",
    "revision": 1
  }
}
```

Step 6: 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:ec2:us-east-1:aws_account_id:task-definition/sleep300:1",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/sleep300:2",
    "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
  ]
}
```
Step 7: Run a Task

After you have registered a task for your account and have launched a container instance that is registered to your cluster, you can run the registered task in your cluster. For this example, you place a single instance of the `sleep360:1` task definition in your default cluster.

```bash
aws ecs run-task --cluster default --task-definition sleep360:1 --count 1
```

Output:

```json
{
  "tasks": [
    {
      "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
      "overrides": {
        "containerOverrides": [
          {
            "name": "sleep"
          }
        ],
        "lastStatus": "PENDING",
        "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID",
        "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/default",
        "desiredStatus": "RUNNING",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/sleep360:1",
        "containers": [
          {
            "containerArn": "arn:aws:ecs:us-east-1:aws_account_id:container/container_ID",
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
            "lastStatus": "PENDING",
            "name": "sleep"
          }
        ]
      }
    }
  ]
}
```

Step 8: List Tasks

List the tasks for your cluster. You should see the task that you ran in the previous section. You can take the task ID or the full ARN that is returned from this command and use it to describe the task later.

```bash
aws ecs list-tasks --cluster default
```

Output:

```json
{
  "taskArns": [
    "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID"
  ]
}
```
Step 9: Describe the Running Task

Describe the task using the task ID retrieved earlier to get more information about the task.

```
aws ecs describe-tasks --cluster default --task task_ID
```

Output:

```
{
  "failures": [],
  "tasks": [
    {
      "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
      "overrides": {
        "containerOverrides": [
          {
            "name": "sleep"
          }
        ],
      },
      "lastStatus": "RUNNING",
      "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID",
      "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/default",
      "desiredStatus": "RUNNING",
      "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/sleep360:1",
      "containers": [
        {
          "containerArn": "arn:aws:ecs:us-east-1:aws_account_id:container/container_ID",
          "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
          "lastStatus": "RUNNING",
          "name": "sleep",
          "networkBindings": []
        }
      ]
    }
  ]
}
```
Common Use Cases in Amazon ECS

This topic provides guidance for two common use cases in Amazon ECS: microservices and batch jobs. Here you can find considerations and external resources that may be useful for getting your application running on Amazon ECS, and the common aspects of each solution.

Topics
- Microservices (p. 567)
- Batch Jobs (p. 569)

Microservices

Microservices are built with a software architectural method that decomposes complex applications into smaller, independent services. Containers are optimal for running small, decoupled services, and they offer the following advantages:

- Containers make services easy to model in an immutable image with all of your dependencies.
- Containers can use any application and any programming language.
- The container image is a versioned artifact, so you can track your container images to the source they came from.
- You can test your containers locally, and deploy the same artifact to scale.

The following sections cover some of the aspects and challenges that you must consider when designing a microservices architecture to run on Amazon ECS. You can also view the microservices reference architecture on GitHub. For more information, see Deploying Microservices with Amazon ECS, AWS CloudFormation, and an Application Load Balancer.

Topics
- Auto Scaling (p. 567)
- Service Discovery (p. 568)
- Authorization and Secrets Management (p. 568)
- Logging (p. 568)
- Continuous Integration and Continuous Deployment (p. 568)

Auto Scaling

The application load for your microservice architecture can change over time. A responsive application can scale out or in, depending on actual or anticipated load. Amazon ECS provides you with several tools to scale not only your services that are running in your clusters, but the actual clusters themselves.

For example, Amazon ECS provides CloudWatch metrics for your clusters and services. For more information, see Amazon ECS CloudWatch Metrics (p. 371). You can monitor the memory and CPU utilization for your clusters and services. Then, use those metrics to trigger CloudWatch alarms that can automatically scale out your cluster when its resources are running low. Scale them back in when you don’t need as many resources. For more information, see Tutorial: Scaling Container Instances with CloudWatch Alarms (p. 379).
In addition to scaling your cluster size, your Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. Service Auto Scaling is available in all regions that support Amazon ECS. For more information, see Service Auto Scaling (p. 321).

**Service Discovery**

Service discovery is a key component of most distributed systems and service-oriented architectures. With service discovery, your microservice components are automatically discovered as they get created and terminated on a given infrastructure. There are several approaches that you can use to make your services discoverable. The following resources describe a few examples:

- **Run Containerized Microservices with Amazon EC2 Container Service and Application Load Balancer:** This post describes how to use the dynamic port mapping and path-based routing features of Elastic Load Balancing Application Load Balancers to provide service discovery for a microservice architecture.
- **Amazon Elastic Container Service - Reference Architecture: Service Discovery:** This Amazon ECS reference architecture provides service discovery to containers using CloudWatch Events, Lambda, and Route 53 private hosted zones.
- **Service Discovery via Consul with Amazon ECS:** This post shows how a third party tool called Consul by HashiCorp can augment the capabilities of Amazon ECS by providing service discovery for an ECS cluster (complete with an example application).

**Authorization and Secrets Management**

Managing secrets, such as database credentials for an application, has always been a challenging issue. The Managing Secrets for Amazon ECS Applications Using Parameter Store and IAM Roles for Tasks post focuses on how to integrate the IAM roles for tasks (p. 435) functionality of Amazon ECS with the AWS Systems Manager Parameter Store. Parameter Store provides a centralized store to manage your configuration data, whether it's plaintext data such as database strings or secrets such as passwords, encrypted through AWS Key Management Service.

**Logging**

You can configure your container instances to send log information to CloudWatch Logs. This enables you to view different logs from your container instances in one convenient location. For more information about getting started using CloudWatch Logs on your container instances that were launched with the Amazon ECS-optimized AMI, see Using CloudWatch Logs with Container Instances (p. 114).

You can configure the containers in your tasks to send log information to CloudWatch Logs. This enables you to 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. For more information about getting started using the awslogs log driver in your task definitions, see Using the awslogs Log Driver (p. 245).

**Continuous Integration and Continuous Deployment**

Continuous integration and continuous deployment (CICD) is a common process for microservice architectures that are based on Docker containers. You can create a pipeline that takes the following actions:

- Monitors changes to a source code repository
- Builds a new Docker image from that source
- Pushes the image to an image repository such as Amazon ECR or Docker Hub
• Updates your Amazon ECS services to use the new image in your application

The following resources outline how to do this in different ways:

• **ECS Reference Architecture: Continuous Deployment**: This reference architecture demonstrates how to achieve continuous deployment of an application to Amazon ECS using CodePipeline, CodeBuild, and AWS CloudFormation.

• **Continuous Delivery Pipeline for Amazon ECS Using Jenkins, GitHub, and Amazon ECR**: This AWS labs repository helps you set up and configure a continuous delivery pipeline for Amazon ECS using Jenkins, GitHub, and Amazon ECR.

• **Pipelines For Container Applications Made Easy with mu**: This post on the AWS Open Source blog shows how to use mu to configure a continuous delivery pipeline for a container workload using Amazon ECS, CodePipeline, and CodeBuild.

**Batch Jobs**

Docker containers are particularly suited for batch job workloads. Batch jobs are often short-lived and embarrassingly parallel. You can package your batch processing application into a Docker image so that you can deploy it anywhere, such as in an Amazon ECS task. If you are interested in running batch job workloads, consider the following resources:

• **AWS Batch**: For fully managed batch processing at any scale, you should consider using AWS Batch. AWS Batch enables developers, scientists, and engineers to easily and efficiently run hundreds of thousands of batch computing jobs on AWS. AWS Batch dynamically provisions the optimal quantity and type of compute resources (for example, CPU or memory-optimized instances) based on the volume and specific resource requirements of the batch jobs submitted. For more information, see the AWS Batch product detail pages.

• **Amazon ECS Reference Architecture: Batch Processing**: This reference architecture illustrates how to use AWS CloudFormation, Amazon S3, Amazon SQS, and CloudWatch alarms to handle batch processing on Amazon ECS.
Getting Started with AWS App Mesh and Amazon ECS

AWS App Mesh is a service mesh based on the Envoy proxy that makes it easy to monitor and control microservices. App Mesh standardizes how your microservices communicate, giving you end-to-end visibility and helping to ensure high-availability for your applications.

App Mesh gives you consistent visibility and network traffic controls for every microservice in an application. For more information, see the AWS App Mesh User Guide.

This topic helps you to use AWS App Mesh with an existing microservice application running on Amazon ECS.

Prerequisites

App Mesh supports microservice applications that use service discovery naming for their components. To use this getting started guide, you must have a microservice application running on Amazon ECS that already has service discovery configured.

For more information about service discovery on Amazon ECS, see Service Discovery (p. 328).

Step 1: Create Your Service Mesh

A service mesh is a logical boundary for network traffic between the services that reside within it. For more information, see Service Meshes in the AWS App Mesh User Guide.

After you create your service mesh, you can create virtual services, virtual nodes, virtual routers, and routes to distribute traffic between the applications in your mesh.

To create a new service mesh with the AWS Management Console

2. Choose Create mesh.
3. For Mesh name, specify a name for your service mesh.
4. Choose Create mesh to finish.

Step 2: Create Your Virtual Nodes

A virtual node acts as a logical pointer to a particular task group, such as a Kubernetes deployment. For more information, see Virtual Nodes in the AWS App Mesh User Guide.

When you create a virtual node, you must specify the DNS service discovery hostname for your task group. Any inbound traffic that your virtual node expects should be specified as a listener. Any outbound traffic that your virtual node expects to reach should be specified as a backend.

You must create virtual nodes for each microservice in your application.
To create a virtual node in the AWS Management Console.

1. Choose the mesh that you created in the previous steps.
2. Choose Virtual nodes in the left navigation.
3. Choose Create virtual node.
4. For Virtual node name, choose a name for your virtual node.
5. For Service discovery method, choose DNS for services that use DNS service discovery and then specify the hostname for DNS hostname. Otherwise, choose None if your virtual node doesn't expect any ingress traffic.
6. To specify any backends (for egress traffic) for your virtual node, or to configure inbound and outbound access logging information, choose Additional configuration.
   a. To specify a backend, choose Add backend and enter a virtual service name or full Amazon Resource Name (ARN) for the virtual service that your virtual node communicates with. Repeat this step until all of your virtual node backends are accounted for.
   b. To configure logging, enter the HTTP access logs path that you want Envoy to use. We recommend the /dev/stdout path so that you can use Docker log drivers to export your Envoy logs to a service such as Amazon CloudWatch Logs.
      
      Note
      Logs must still be ingested by an agent in your application and sent to a destination. This file path only instructs Envoy where to send the logs.
7. If your virtual node expects ingress traffic, specify a Port and Protocol for that Listener.
8. If you want to configure health checks for your listener, ensure that Health check enabled is selected and then complete the following substeps. If not, clear this check box.
   a. For Health check protocol, choose to use an HTTP or TCP health check.
   b. For Health check port, specify the port that the health check should run on.
   c. For Healthy threshold, specify the number of consecutive successful health checks that must occur before declaring the listener healthy.
   d. For Health check interval, specify the time period in milliseconds between each health check execution.
   e. For Path, specify the destination path for the health check request. This is required only if the specified protocol is HTTP. If the protocol is TCP, this parameter is ignored.
   f. For Timeout period, specify the amount of time to wait when receiving a response from the health check, in milliseconds.
   g. For Unhealthy threshold, specify the number of consecutive failed health checks that must occur before declaring the listener unhealthy.
9. Chose Create virtual node to finish.
10. Repeat this procedure as necessary to create virtual nodes for each remaining microservice in your application.

Step 3: Create Your Virtual Routers

Virtual routers handle traffic for one or more virtual services within your mesh. After you create a virtual router, you can create and associate routes for your virtual router that direct incoming requests to different virtual nodes. For more information, see Virtual Routers in the AWS App Mesh User Guide.

Create virtual routers for each microservice in your application.

Creating a virtual router in the AWS Management Console.

1. Choose Virtual routers in the left navigation.
2. Choose **Create virtual router**.
3. For **Virtual router name**, specify a name for your virtual router. Up to 255 letters, numbers, hyphens, and underscores are allowed.
4. For **Listener**, specify a **Port** and **Protocol** for your virtual router.
5. Choose **Create virtual router** to finish.
6. Repeat this procedure as necessary to create virtual routers for each remaining microservice in your application.

**Step 4: Create Your Routes**

A route is associated with a virtual router, and it's used to match requests for a virtual router and distribute traffic accordingly to its associated virtual nodes. For more information, see Routes in the AWS App Mesh User Guide.

Create routes for each microservice in your application.

**Creating a route in the AWS Management Console.**

1. Choose **Virtual routers** in the left navigation.
2. Choose the router that you want to associate a new route with.
3. In the **Routes** table, choose **Create route**.
4. For **Route name**, specify the name to use for your route.
5. For **Route type**, choose the protocol for your route.
6. For **Virtual node name**, choose the virtual node that this route will serve traffic to.
7. For **Weight**, choose a relative weight for the route. The total weight for all routes must be less than 100.
8. To use HTTP path-based routing, choose **Additional configuration** and then specify the path that the route should match. For example, if your virtual service name is `my-service.local` and you want the route to match requests to `my-service.local/metrics`, your prefix should be `/metrics`.
9. Choose **Create route** to finish.
10. Repeat this procedure as necessary to create routes for each remaining microservice in your application.

**Step 5: Create Your Virtual Services**

A virtual service is an abstraction of a real service that is provided by a virtual node directly or indirectly by means of a virtual router. Dependent services call your virtual service by its **virtualServiceName**, and those requests are routed to the virtual node or virtual router that is specified as the provider for the virtual service. For more information, see Virtual Services in the AWS App Mesh User Guide.

Create virtual services for each microservice in your application.

**Creating a virtual service in the AWS Management Console.**

1. Choose **Virtual services** in the left navigation.
2. Choose **Create virtual service**.
3. For **Virtual service name**, choose a name for your virtual service. We recommend that you use the service discovery name of the real service that you're targeting (such as `my-service.default.svc.cluster.local`).
4. For Provider, choose the provider type for your virtual service:
   - If you want the virtual service to spread traffic across multiple virtual nodes, select Virtual router and then choose the virtual router to use from the drop-down menu.
   - If you want the virtual service to reach a virtual node directly, without a virtual router, select Virtual node and then choose the virtual node to use from the drop-down menu.
   - If you don't want the virtual service to route traffic at this time (for example, if your virtual nodes or virtual router doesn't exist yet), choose None. You can update the provider for this virtual service later.

5. Choose Create virtual service to finish.

6. Repeat this procedure as necessary to create virtual services for each remaining microservice in your application.

---

Update Your Microservice Task Definitions

After you create your service mesh, virtual nodes, virtual routers, routes, and virtual services, you must update the Amazon ECS task definitions for your microservices to be compatible with App Mesh. Complete the steps in the following sections to update your services' task definitions to work with App Mesh. When you are finished, update your Amazon ECS services to start using App Mesh with your Amazon ECS application.

Proxy Configuration

To configure your Amazon ECS service to use App Mesh, your service's task definition must have the following proxy configuration section. Set the proxy configuration type to APPMESH and the containerName to envoy. Set the following property values accordingly.

- **IgnoredUID**
  
  Envoy doesn't proxy traffic from processes that use this user ID. You can choose any user ID that you want for this (our examples use 1337 for historical purposes), but this ID must be the same as the user ID for the Envoy container in your task definition. This matching allows Envoy to ignore its own traffic without using the proxy.

- **ProxyIngressPort**
  
  This is the ingress port for the Envoy proxy container. Set this value to 15000.

- **ProxyEgressPort**
  
  This is the egress port for the Envoy proxy container. Set this value to 15001.

- **AppPorts**
  
  Specify any ingress ports that your application containers listen on. In this example, the application container listens on port 9080.

- **EgressIgnoredIPs**
  
  Envoy doesn't proxy traffic to these IP addresses. Set this value to 169.254.170.2, 169.254.169.254, which ignores the Amazon EC2 metadata server and the Amazon ECS task metadata endpoint (which provides IAM roles for tasks credentials).

```json
"proxyConfiguration": {
  "type": "APPMESH",
  "containerName": "envoy",
  "properties": [
    {
      "IgnoredUID": "1337",
      "ProxyIngressPort": 15000,
      "ProxyEgressPort": 15001,
      "AppPorts": [9080],
      "EgressIgnoredIPs": ["169.254.170.2", "169.254.169.254"]
    }
  ]
}
```
Application Container Envoy Dependency

The application containers in your task definitions must wait for the Envoy proxy to bootstrap and start before they can start. To ensure that this happens, you set a `dependsOn` section in each application container definition to wait for the Envoy container to report as `HEALTHY`. The following code block shows an application container definition example with this dependency.

```json
{
    "name": "app",
    "image": "application_image",
    "portMappings": [
        {
            "containerPort": 9080,
            "hostPort": 9080,
            "protocol": "tcp"
        }
    ],
    "essential": true,
    "dependsOn": [
        {
            "containerName": "envoy",
            "condition": "HEALTHY"
        }
    ]
}
```

Envoy Container Definition

Your Amazon ECS services' task definitions must contain the App Mesh custom Envoy container image.

111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod

The Envoy container definition must be marked as `essential`. The virtual node name for the Amazon ECS service should be set to the `APP_MESH_VIRTUAL_NODE_NAME`, and the `user ID` value should match the `IgnoredUID` value from the task definition proxy configuration (in this example, we use `1337`).

The health check shown here waits for the Envoy container to bootstrap properly before reporting to Amazon ECS that it is healthy and ready for the application containers to start.
The following code block shows an Envoy container definition example.

```json
{
  "name": "envoy",
  "image": "111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod",
  "essential": true,
  "environment": [
    {
      "name": "APPMESH_VIRTUAL_NODE_NAME",
      "value": "mesh/meshName/virtualNode/virtualnodeName"
    }
  ],
  "healthCheck": {
    "command": [
      "CMD-SHELL",
      "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
    ],
    "startPeriod": 10,
    "interval": 5,
    "timeout": 2,
    "retries": 3
  },
  "user": "1337"
}
```

Example Task Definition

The following example Amazon ECS task definition shows in context the snippets that you can merge with your existing task groups. Substitute your mesh name and virtual node name for the `APPMESH_VIRTUAL_NODE_NAME` value and a list of ports that your application listens on for the proxy configuration `AppPorts` value.

Example JSON for Amazon ECS task definition

```json
{
  "family": "appmesh-gateway",
  "memory": "256",
  "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"
      }
    ]
  }
}
```
Example Task Definition

```
{
  "containerDefinitions": [
    {
      "name": "app",
      "image": "application_image",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
      "name": "envoy",
      "image": "111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/meshName/virtualNode/virtualNodeName"
        }
      ],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
        ],
        "startPeriod": 10,
        "interval": 5,
        "timeout": 2,
        "retries": 3
      },
      "user": "1337"
    }
  ],
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```
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.

To get started using Deep Learning Containers on Amazon ECS, see Deep Learning Containers on Amazon ECS in the AWS Deep Learning AMI Developer Guide.
Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters

Container instances in your clusters need external network access to communicate with the Amazon ECS service endpoint. However, you might have tasks and services that you would like to run in private subnets. Creating a VPC with both public and private subnets provides you the flexibility to launch tasks and services in either a public or private subnet. Tasks and services in the private subnets can access the internet through a NAT gateway. Services in both the public and private subnets can be configured to use a load balancer so that they can still be reached from the public internet.

This tutorial guides you through creating a VPC with two public subnets and two private subnets, which are provided with internet access through a NAT gateway.

Topics
- Step 1: Create an Elastic IP Address for Your NAT Gateway (p. 578)
- Step 2: Run the VPC Wizard (p. 578)
- Step 3: Create Additional Subnets (p. 579)
- Next Steps (p. 579)

Step 1: Create an Elastic IP Address for Your NAT Gateway

A NAT gateway requires an Elastic IP address in your public subnet, but the VPC wizard does not create one for you. Create the Elastic IP address before running the VPC wizard.

To create an Elastic IP address
1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, choose Elastic IPs.
3. Choose Allocate new address, Allocate, Close.
4. Note the Allocation ID for your newly created Elastic IP address; you enter this later in the VPC wizard.

Step 2: Run the VPC Wizard

The VPC wizard automatically creates and configures most of your VPC resources for you.

To run the VPC wizard
1. In the left navigation pane, choose VPC Dashboard.
2. Choose Start VPC Wizard, VPC with Public and Private Subnets, Select.
3. For **VPC name**, give your VPC a unique name.
4. For **Elastic IP Allocation ID**, choose the ID of the Elastic IP address that you created earlier.
5. Choose **Create VPC**.
6. When the wizard is finished, choose **OK**. Note the Availability Zone in which your VPC subnets were created. Your additional subnets should be created in a different Availability Zone.

### Step 3: Create Additional Subnets

The wizard creates a VPC with a single public and a single private subnet in a single Availability Zone. For greater availability, you should create at least one more of each subnet type in a different Availability Zone so that your VPC has both public and private subnets across two Availability Zones.

**To create an additional private subnet**

1. In the left navigation pane, choose **Subnets**.
2. Choose **Create Subnet**.
3. For **Name tag**, enter a name for your subnet, such as **Private subnet**.
4. For **VPC**, choose the VPC that you created earlier.
5. For **Availability Zone**, choose a different Availability Zone than your original subnets in the VPC.
6. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.3.0/24** for your second private subnet.
7. Choose **Yes, Create**.

**To create an additional public subnet**

1. In the left navigation pane, choose **Subnets** and then **Create Subnet**.
2. For **Name tag**, enter a name for your subnet, such as **Public subnet**.
3. For **VPC**, choose the VPC that you created earlier.
4. For **Availability Zone**, choose the same Availability Zone as the additional private subnet that you created in the previous procedure.
5. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.2.0/24** for your second public subnet.
6. Choose **Yes, Create**.
7. Select the public subnet that you just created and choose **Route Table, Edit**.
8. By default, the private route table is selected. Choose the other available route table so that the 0.0.0.0/0 destination is routed to the internet gateway (**igw-xxxxxxxx**) and choose **Save**.
9. With your second public subnet still selected, choose **Subnet Actions, Modify auto-assign IP settings**.
10. Select **Enable auto-assign public IPv4 address** and choose **Save, Close**.

### Next Steps

After you have created your VPC, you should consider the following next steps:

- Create security groups for your public and private resources if they require inbound network access. For more information, see **Working with Security Groups** in the *Amazon VPC User Guide*.
- Create Amazon ECS clusters in your private or public subnets. For more information, see **Creating a Cluster (p. 55)**. If you use the cluster creation wizard in the Amazon ECS console, you can specify
the VPC that you just created and the public or private subnets in which to launch your instances, depending on your use case.

- To make your containers directly accessible from the internet, launch instances into your public subnets. Be sure to configure your container instance security groups appropriately.
- To avoid making containers directly accessible from the internet, launch instances into your private subnets.
- Create a load balancer in your public subnets that can route traffic to services in your public or private subnets. For more information, see Service Load Balancing (p. 306).
Tutorial: Using Amazon EFS File Systems with Amazon ECS

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with Amazon EC2 instances. With Amazon EFS, storage capacity is elastic, growing and shrinking automatically as you add and remove files. Your applications can have the storage they need, 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. However, you must configure your container instance AMI to mount the Amazon EFS file system before the Docker daemon starts. Also, your task definitions must reference volume mounts on the container instance to use the file system. The following sections help you get started using Amazon EFS with Amazon ECS.

**Note**
Amazon EFS is not available in all regions. For more information about which regions support Amazon EFS, see Amazon Elastic File System in the AWS Regions and Endpoints section of the AWS General Reference.

**Topics**
- Step 1: Gather Cluster Information (p. 581)
- Step 2: Create a Security Group for an Amazon EFS File System (p. 582)
- Step 3: Create an Amazon EFS File System (p. 582)
- Step 4: Configure Container Instances (p. 583)
- Step 5: Create a Task Definition to Use the Amazon EFS File System (p. 584)
- Step 6: Add Content to the Amazon EFS File System (p. 585)
- Step 7: Run a Task and View the Results (p. 586)

### Step 1: Gather Cluster Information

Before you can create all of the required resources to use Amazon EFS with your Amazon ECS cluster, gather some basic information about the cluster, such as the VPC it is hosted inside of, and the security group that it uses.

**To gather the VPC and security group IDs for a cluster**

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Select one of the container instances from your cluster and view the **Description** tab of the instance details. If you created your cluster with the Amazon ECS first-run or cluster creation wizards, the cluster name should be part of the EC2 instance name. For example, a cluster named `default` has this EC2 instance name: `ECS_Instance - EC2ContainerService-default`.
3. Record the **VPC ID** value for your container instance. Later, you create a security group and an Amazon EFS file system in this VPC.
4. Open the security group to view its details.
Step 2: Create a Security Group for an Amazon EFS File System

In this section, you create a security group for your Amazon EFS file system that allows inbound access from your container instances.

To create a security group for an Amazon EFS file system

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the left navigation pane, choose Security Groups, Create Security Group.
3. For Security group name, enter a unique name for your security group. For example, EFS-access-for-sg-025fa2.
4. For Description, enter a description for your security group.
5. For VPC, choose the VPC that you identified earlier for your cluster.
6. Choose Inbound, Add rule.
7. For Type, choose All traffic.
8. For Source, choose Custom and then enter the security group ID that you identified earlier for your cluster.
9. Choose Create.

Step 3: Create an Amazon EFS File System

Before you can use Amazon EFS with your container instances, you must create an Amazon EFS file system.

To create an Amazon EFS file system for Amazon ECS container instances

   **Note**
   Amazon EFS is not available in all regions. For more information about which regions support Amazon EFS, see Amazon Elastic File System in the AWS Regions and Endpoints section of the AWS General Reference.
2. Choose Create file system.
3. On the Configure file system access page, choose the VPC that your container instances are hosted in and choose Next Step. By default, each subnet in the specified VPC receives a mount target that uses the default security group for that VPC.
   **Note**
   Your Amazon EFS file system and your container instances must be in the same VPC.
4. For Security groups, add the security group that you created in the previous section. Choose Next step.
5. (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.
6. Choose a performance mode for your file system and choose Next Step.
Step 4: Configure Container Instances

After you've created your Amazon EFS file system in the same VPC as your container instances, you must configure the container instances to access and use the file system.

Configure a running container instance to use an Amazon EFS file system

1. Log in to the container instance via SSH. For more information, see Connect to Your Container Instance (p. 113).
2. Create a mount point for your Amazon EFS file system. For example, `/efs`.

   ```bash
   sudo mkdir /efs
   ```

3. Install NFS client software on your container instance.
   - For Amazon Linux, CentOS, and Red Hat Enterprise Linux:
     ```bash
     sudo yum install -y nfs-utils
     ```
   - For Ubuntu and Debian:
     ```bash
     sudo apt-get install -y nfs-common
     ```

4. Mount your file system with the following command. Be sure to replace the file system ID and region with your own.

   ```bash
   sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 fs-613c8628.efs.us-east-1.amazonaws.com:/efs
   ```

5. Validate that the file system is mounted correctly with the following command. You should see a file system entry that matches your Amazon EFS file system. If not, see Troubleshooting Amazon EFS in the Amazon Elastic File System User Guide.

   ```bash
   mount | grep efs
   ```

6. Make a backup of the `/etc/fstab` file.

   ```bash
   sudo cp /etc/fstab /etc/fstab.bak
   ```

7. Update the `/etc/fstab` file to automatically mount the file system at boot.

   ```bash
   echo 'fs-613c8628.efs.us-east-1.amazonaws.com:/efs nfs4
efsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 0 0' | sudo tee -a /etc/fstab
   ```

8. Reload the file system table to verify that your mounts are working properly.

   ```bash
   sudo mount -a
   ```
Step 5: Create a Task Definition to Use the Amazon EFS File System

Because the file system is mounted on the host container instance, you must create a volume mount in your Amazon ECS task definition that allows your containers to access the file system. For more information, see Using Data Volumes in Tasks (p. 229).

The following task definition creates a data volume called efs-html at /efs/html on the host container instance Amazon EFS file system. The nginx container mounts the host data volume at the NGINX root, /usr/share/nginx/html.

```json
{
}
```
Step 6: Add Content to the Amazon EFS File System

For the NGINX example task, you created a directory at /efs/html on the container instance to host the web content. Before the NGINX containers can serve any web content, you must add the content to the file system. In this section, you log in to a container instance and add an index.html file.

To add content to the file system

1. Connect using SSH to one of your container instances that is using the Amazon EFS file system. For more information, see Connect to Your Container Instance (p. 113).
2. Write a simple HTML file by copying and pasting the following block of text into a terminal.

```bash
sudo bash -c "cat >/efs/html/index.html" <<'EOF'
<html>
  <body>
    <h1>It Works!</h1>
    <p>You are using an Amazon EFS file system for persistent container storage.</p>
  </body>
EOF"
```

You can save this task definition to a file called nginx-efs.json and register it to use in your own clusters with the following AWS CLI command. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide.

```bash
aws ecs register-task-definition --cli-input-json file://nginx-efs.json
```
Step 7: Run a Task and View the Results

Now that your Amazon EFS file system is available on your container instances and there is web content for the NGINX containers to serve, you can run a task using the task definition that you created earlier. The NGINX web servers serve 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.

To run a task and view the results

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose the cluster that you have configured to use Amazon EFS.
3. Choose **Tasks, Run new task**.
4. For **Task Definition**, choose the `nginx-efs` task definition that you created earlier and choose **Run Task**. For more information on the other options in the run task workflow, see Running Tasks (p. 269).
5. Below the **Tasks** tab, choose the task that you just ran.
6. Expand the container name at the bottom of the page, and choose the IP address that is associated with the container. Your browser should open a new tab with 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 instances allows inbound network traffic on port 80.
Amazon ECS Service Limits

The following table provides the default limits for Amazon ECS for an AWS account which can be changed. For more information on the service limits for other AWS services that you can use with Amazon ECS, such as Elastic Load Balancing and Auto Scaling, see AWS Service Limits in the Amazon Web Services General Reference.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of clusters per Region, per account</td>
<td>2000</td>
</tr>
<tr>
<td>Number of container instances per cluster</td>
<td>2000</td>
</tr>
<tr>
<td>Number of services per cluster</td>
<td>1000</td>
</tr>
<tr>
<td>Number of tasks per service (the desired count)</td>
<td>1000</td>
</tr>
<tr>
<td>Number of tasks using the Fargate launch type, per Region, per account</td>
<td>50</td>
</tr>
<tr>
<td>Number of public IP addresses for tasks using the Fargate launch type, per Region, per account</td>
<td>50</td>
</tr>
</tbody>
</table>

The following table provides other limitations for Amazon ECS that cannot be changed.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of load balancers per service</td>
<td>1</td>
</tr>
<tr>
<td>Number of tasks launched (count) per run-task</td>
<td>10</td>
</tr>
<tr>
<td>Number of container instances per start-task</td>
<td>10</td>
</tr>
<tr>
<td>Number of revisions per task definition family</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

**Note**

Deregistering a task definition revision does not exclude it from being included in this limit.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task definition size limit</td>
<td>32 KiB</td>
</tr>
<tr>
<td>Task definition max containers</td>
<td>10</td>
</tr>
<tr>
<td>Number of subnets specified in awsvpcConfiguration</td>
<td>16</td>
</tr>
<tr>
<td>Number of security groups specified in awsvpcConfiguration</td>
<td>5</td>
</tr>
<tr>
<td>Maximum size of a shared volume used by multiple containers within a task using the Fargate launch type</td>
<td>4 GB</td>
</tr>
<tr>
<td>Resource</td>
<td>Limit</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Maximum container storage for tasks using the Fargate launch type</td>
<td>10 GB</td>
</tr>
<tr>
<td>Maximum number of tags per resource (tasks, services, task definitions, clusters, and container instances)</td>
<td>50</td>
</tr>
</tbody>
</table>
Amazon ECS Troubleshooting

You may 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.

Topics
- Troubleshooting First-Run Wizard Launch Issues (p. 589)
- Checking Stopped Tasks for Errors (p. 590)
- Service Event Messages (p. 591)
- Invalid CPU or Memory Value Specified (p. 594)
- Cannot Pull Container Image Error (p. 595)
- CannotCreateContainerError: API error (500): devmapper (p. 596)
- Troubleshooting Service Load Balancers (p. 597)
- Enabling Docker Debug Output (p. 599)
- Amazon ECS Log File Locations (p. 600)
- Amazon ECS Logs Collector (p. 601)
- Agent Introspection Diagnostics (p. 602)
- Docker Diagnostics (p. 604)
- API failures Error Messages (p. 606)
- Troubleshooting IAM Roles for Tasks (p. 607)

Troubleshooting First-Run Wizard Launch Issues

The following error can prevent the Amazon ECS first-run wizard from creating your cluster.

VpcLimitExceeded

You may get a VpcLimitExceeded error when attempting to complete the Amazon ECS first-run wizard. If so, you have reached the limit on the number of VPCs that you can create in a Region. When you create your AWS account, there are default limits on the number of VPCs that you can run in each Region. For more information, see Amazon VPC Limits.

To resolve this issue, you have the following options:
- Request a VPC service limit increase on a per-Region basis. For more information, see Amazon VPC Limits.
- Delete any unused VPCs on your account. For more information, see Working with VPCs and Subnets.

Important
Any Amazon ECS resources that were successfully created during the first-run wizard before receiving this error can be deleted before running the wizard again.
Checking Stopped Tasks for Errors

If you have trouble starting a task, your task might be stopping because of an error. For example, you run the task and the task displays a PENDING status and then disappears. You can view errors like this in the Amazon ECS console by displaying the stopped task and inspecting it for error messages.

To check stopped tasks for errors

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the Clusters page, select the cluster in which your stopped task resides.
4. In the Desired task status table header, choose Stopped, and then select the stopped task to inspect. The most recent stopped tasks are listed first.
5. In the Details section, inspect the Stopped reason field to see the reason that the task was stopped.

Some possible reasons and their explanations are listed below:

Task failed ELB health checks in (elb elb-name)

The current task failed the Elastic Load Balancing health check for the load balancer that is associated with the task's service. For more information, see Troubleshooting Service Load Balancers (p. 597).

Scaling activity initiated by (deployment deployment-id)

When you reduce the desired count of a stable service, some tasks must be stopped in order to reach the desired number. Tasks that are stopped by downscaling services have this stopped reason.

Host EC2 (instance id) stopped/terminated

If you stop or terminate a container instance with running tasks, then the tasks are given this stopped reason.
Container instance deregistration forced by user

If you force the deregistration of a container instance with running tasks, then the tasks are given this stopped reason.

Essential container in task exited

If a container marked as essential in task definitions exits or dies, that can cause a task to stop. When an essential container exiting is the cause of a stopped task, the Step 6 (p. 591) can provide more diagnostic information as to why the container stopped.

6. If you have a container that has stopped, expand the container and inspect the Status reason row to see what caused the task state to change.

Containers

<table>
<thead>
<tr>
<th>Name</th>
<th>Container Id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>curler</td>
<td>3f871451-c9f1-4d6f-a...</td>
<td>STOPPED (CannotPullContainerError: Error: image tutum/bogus)</td>
</tr>
</tbody>
</table>

In the previous example, the container image name cannot be found. This can happen if you misspell the image name.

If this inspection does not provide enough information, you can connect to the container instance with SSH and inspect the Docker container locally. For more information, see Inspect Docker Containers (p. 605).

Service Event Messages

If you are troubleshooting a problem with a service, the first place you should check for diagnostic information is the service event log.

To check the service event log in the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the Clusters page, select the cluster in which your service resides.
3. On the Cluster: clustername page, select the service to inspect.
5. Examine the **Message** column for errors or other helpful information.

### Service Event Messages

The following are examples of service event messages you may see in the console:

- (service **service-name**) was unable to place a task because the resources could not be found. (p. 592)
- (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". (p. 593)
- (service **service-name**) (**instance** **instance-id**) is unhealthy in (**elb** **elb-name**) due to (reason **Instance has failed at least the UnhealthyThreshold number of health checks consecutively.**) (p. 594)
- (service **service-name**) is unable to consistently start tasks successfully. (p. 594)

(service **service-name**) was unable to place a task because the resources could not be found.

In the above image, this service could not find the available resources to add another task. The possible causes for this are:

**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.

**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 Container Instance Memory Management (p. 124).

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, which is attached to the container instance that hosts it. 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 example, a c4.large instance may have three network interfaces attached to it. The primary network adapter for the instance counts as one, so you can attach two more ENIs to the instance. Because each awsvpc task requires a network interface, you can only run two such tasks on this instance type. For more information about how many network interfaces are supported per instance type, see IP Addresses Per Network Interface Per Instance Type in the Amazon EC2 User Guide for Linux Instances. 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 do not have any available container instances that can satisfy this requirement, the task cannot be placed. For more information on which attributes are required for specific task definition parameters and agent configuration variables, see Task Definition Parameters (p. 195) and Amazon ECS Container Agent Configuration (p. 152).

Note
Windows container instances with Amazon ECS container agent versions earlier than 1.17.0 do not support the awslogs log driver by default. If you are unable to use the awslogs log driver with your Windows container instances, ensure that you are using the latest Amazon ECS-optimized Windows AMI.

(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 (p. 600). 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:

```bash
sudo systemctl restart ecs
```
- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs && sudo start ecs
```

**(service service-name) (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. For more information, see Troubleshooting Service Load Balancers (p. 597).

**(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 Service Throttle Logic (p. 354).

After the service is updated, for example with an updated task definition, the service scheduler resumes normal behavior.

### Invalid CPU or Memory Value Specified

When registering a task, if you specify an invalid `cpu` or `memory` value, you receive the following error:

```
An error occurred (ClientException) when calling the RegisterTaskDefinition operation: Invalid 'cpu' setting for task. For more information, see the Troubleshooting section of the Amazon ECS Developer Guide.
```

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 but is converted to an integer indicating the CPU units when the task definition is registered. If you are using the EC2 launch type, the supported values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs). If you are using the Fargate launch type, you must use one of the values in the following table, which determines your range of supported values for the `memory` parameter.

The `memory` value can be expressed in MiB or GB in a task definition but is converted to an integer indicating the MiB when the task definition is registered. If you are using the EC2 launch type, you must specify an integer. If you are using the Fargate launch type, you must use one of the values in the following table, which determines your range of supported values for the `cpu` parameter.

Supported task CPU and memory values for Fargate tasks are as follows.

<table>
<thead>
<tr>
<th>CPU value</th>
<th>Memory value (MiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 (.25 vCPU)</td>
<td>512 (0.5GB), 1024 (1GB), 2048 (2GB)</td>
</tr>
<tr>
<td>512 (.5 vCPU)</td>
<td>1024 (1GB), 2048 (2GB), 3072 (3GB), 4096 (4GB)</td>
</tr>
<tr>
<td>1024 (1 vCPU)</td>
<td>2048 (2GB), 3072 (3GB), 4096 (4GB), 5120 (5GB), 6144 (6GB), 7168 (7GB), 8192 (8GB)</td>
</tr>
</tbody>
</table>
Cannot Pull Container Image Error

The following Docker errors indicate that when creating a task, the container image specified could not be retrieved.

Connection timed out

When a Fargate task is launched, its elastic network interface requires a route to the internet to pull container images. If you receive an error similar to the following when launching a task, it is because a route to the internet does not 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 Tasks (p. 269).
- 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. For more information about creating a VPC with public and private subnets, including a NAT gateway for the private subnets, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters (p. 578).

Image not found

When you specify an Amazon ECR image in your container definition, you must use the full ARN or URI of your ECR repository along with the image name in that repository. If the repository or image cannot be found, you receive the following error:

```
CannotPullContainerError: API error (404): repository 111122223333.dkr.ecr.us-east-1.amazonaws.com/<repo>/<image> not found
```

To resolve this issue, verify the repository ARN or URI and the image name. Also ensure that you have set up the proper access using the task execution IAM role. For more information about the task execution role, see Amazon ECS Task Execution IAM Role (p. 418).

Insufficient disk space

If the root volume of your container instance has insufficient disk space when pulling the container image, you see an error similar to the following:

```
CannotPullContainerError: write /var/lib/docker/tmp/GetImageBlob1111111111: no space left on device
```

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 filesystem:
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/
c8e3010e36c089bf286a623699f5233097ca126ed5a700af023a5127633d/rootfs/data/logs
...

In some cases, like this example above, the root volume may 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:

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>50501b5f4cbf</td>
<td>amazon/amazon-ecs-agent:latest</td>
<td>&quot;/agent&quot;</td>
<td>4 days ago</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Up 4 days</td>
<td>ecs-agent</td>
</tr>
</tbody>
</table>

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 are able to set the max-size as a log driver option, which prevents the log file from taking up too much space. For more information, see Configure logging drivers in the Docker documentation.

The following is a container definition snippet showing how to use this option:

```json
{
   "log-driver": "json-file",
   "log-opt": {
      "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 Using the awslogs Log Driver (p. 245).

---

**CannotCreateContainerError: API error (500): devmapper**

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 2015.09.0 and later launch with an 8-GiB volume for the operating system that is attached at /dev/xvda and mounted as the root of the file system. There is an additional 22-GiB volume that is 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 are unable to do this, you can add storage to the volume group that Docker uses and extend its logical volume by following the procedures in AMI Storage Configuration (p. 87).

If your container instance storage is filling up too quickly, there are a few actions that you can take to reduce this effect:

- (Amazon ECS container agent 1.8.0 and later) 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 may 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 (p. 152).

- 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 cannot be inspected later, and deleted images must be pulled again before starting new containers from them.

To remove non-running containers, execute the following command on your container instance:

```
docker rm $(docker ps -aq)
```

To remove unused images, execute the following command on your container instance:

```
docker rmi $(docker images -q)
```

- 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/"
```

Troubleshooting Service Load Balancers

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.

**Important**

Container health checks are not supported for tasks that are part of a service that is configured to use a Classic Load Balancer. The Amazon ECS service scheduler ignores tasks in an UNHEALTHY state that are behind a Classic Load Balancer.

**Improper IAM permissions for the ecsServiceRole IAM role**

The ecsServiceRole allows Amazon ECS services to register container instances with Elastic Load Balancing load balancers. You must have the proper permissions set for this role. For more information, see Amazon ECS Service Scheduler IAM Role (p. 426).
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 in which 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 is not configured to use, the task never passes the health check and it is killed.

Elastic Load Balancing load balancer health check misconfigured

The load balancer health check parameters can be overly restrictive or point to resources that do not exist. If a container instance is determined to be unhealthy, it is 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 are using with the health check.

Ping Path

This value is often set to index.html, but if your service does not respond to that request, then the health check fails. If your container does not 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 is 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, the load balancer configuration defined for your service when it was created cannot be changed. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

To change the load balancer name, the container name, or the container port associated with a service load balancer configuration, you must create a new service.
Enabling Docker Debug Output

If you are having trouble with Docker containers or images, you can enable debug mode on your Docker daemon. Enabling debugging provides more verbose output from the daemon and you can use this information to find out more about why your containers or images are having issues.

Enabling Docker debug mode can be especially useful in retrieving error messages that are sent from container registries, such as Amazon ECR, and, in many circumstances, enabling debug mode is the only way to see these error messages.

**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 enable Docker daemon debug mode on the Amazon ECS-optimized Amazon Linux AMI**

1. Connect to your container instance. For more information, see Connect to Your Container Instance (p. 113).
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
   ```

   Output:

   ```
   Stopping docker: [ OK ]
   Starting docker: [ OK ]
   ```

6. Restart the Amazon ECS agent.

   ```
   sudo start ecs
   ```

   Your Docker logs should now show more verbose output. For example:

   ```
   ```
Amazon ECS Log File Locations

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 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. For more information, see Connect to Your Container Instance (p. 113).

**Note**
If you are unsure how to collect all of the various logs on your container instances, you can use the Amazon ECS logs collector. For more information, see Amazon ECS Logs Collector (p. 601).

Amazon ECS Container Agent Log

The Amazon ECS container agent stores logs at /var/log/ecs/ecs-agent.log.timestamp on Linux instances, and C:\ProgramData\Amazon\ECS\log\ecs-agent.log.timestamp on Windows instances.

**Note**
You can increase the verbosity of the container agent logs by setting ECS_LOGLEVEL=debug and restarting the container agent. For more information, see Amazon ECS Container Agent Configuration (p. 152).

```bash
cat /var/log/ecs/ecs-agent.log.2016-08-15-15
```

**Output:**

```
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] Registered! module="api client"
```

Amazon ECS ecs-init Log

The ecs-init process stores logs at /var/log/ecs/ecs-init.log.

```bash
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
```
IAM Roles for Tasks Credential Audit Log

When the credential provider for the IAM role is used to provide credentials to tasks, these requests are logged in /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

An example log entry is shown below:

```
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
```

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
- Windows 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 does not 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. For more information, see Connect to Your Container Instance (p. 113).
2. Download the Amazon ECS logs collector script.
   
   ```shell
   ```
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=debug` option to the command below. This may 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 Container Instance Draining (p. 122).
   
   ```shell
   [ec2-user ~]$ sudo bash ./ecs-logs-collector.sh
   ```

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 Connecting to Your Windows Instance in the Amazon EC2 User Guide for Windows Instances.
2. Download the Amazon ECS logs collector script using PowerShell.
   
   ```powershell
   ```
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 command below. 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 Container Instance Draining (p. 122).
   
   ```powershell
   .\ecs-logs-collector.ps1
   ```

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.

Agent Introspection Diagnostics

The Amazon ECS agent introspection API can provide helpful diagnostic information. For example, 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. For more information, see Connect to Your Container Instance (p. 113).
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 (p. 415).

The below example shows two tasks, one that is currently running and one that was stopped.

Note
The command below is piped through the `python -m json.tool` for greater readability.

curl http://localhost:51678/v1/tasks | python -m json.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": "189a8ff4b5f04affe40e5160a5ffadca395136eb5f0f4950c57963c06f82c76d",
       "DockerName": "ecs-console-sample-app-static-6-simple-app-86caf9bcafe3e9c61600",
       "Name": "simple-app"
      },
      { "DockerId": "f7f1f8a7a245c5da83aa92729d28c8b004d1f6a35409e4207e1d34030e966",
       "DockerName": "ecs-console-sample-app-static-6-busybox-ce83ce978a87890a01",
       "Name": "busybox"
      }
   ],
   "Family": "console-sample-app-static",
   "KnownStatus": "STOPPED",
   "Version": "6"
   },
    "Containers": [
      { "DockerId": "dc7240fe892ab233dbce5046d95e1456c120dbab9a6b56ec513da45c38e3aeb",
       "DockerName": "ecs-console-sample-app-static-6-simple-app-f0e5859699a7aeccfb101",
       "Name": "simple-app"
      },
      { "DockerId": "096d685f85a1ff3e021c8254672ab8497e3c13986b9cf005cbea9460b7b901e",
       "DockerName": "ecs-console-sample-app-static-6-busybox-92e4b8d0edc0ccee69a01",
       "Name": "busybox"
      }
   ],
   "DesiredStatus": "RUNNING",
   "Family": "console-sample-app-static",
   "Version": "6"
]
```
Amazon Elastic Container Service Developer Guide

Docker Diagnostics

Docker Diagnostics

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 Command Line topic in the Docker documentation. You can access the Docker command line utilities by connecting to a container instance using SSH. For more information, see Connect to Your Container Instance (p. 113).

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

You can use the docker ps command on your container instance to list the running containers. In the below example, only the Amazon ECS container agent is running. For more information, see docker ps in the Docker documentation.

docker ps

Output:

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>cee0d6986de0</td>
<td>amazon/amazon-ecs-agent:latest</td>
<td>/agent</td>
<td>22 hours ago</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 f7ff1f8a7a245 exited 9 seconds ago, so it does not show up in a docker ps output without the -a flag.

docker ps -a

Output:

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>db4d48e411b1</td>
<td>amazon/ecs-emptyvolume-base:autogenerated</td>
<td>&quot;not-applicable&quot;</td>
<td>19 seconds ago</td>
</tr>
<tr>
<td></td>
<td>sample-app-static-6-internalecs-emptyvolume-source-c09288a6b0c8a53700</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f7ff1f8a7a245</td>
<td>busybox:buildroot-2014.02</td>
<td>&quot;&quot;sh -c '/bin/sh -c 22 hours ago</td>
<td>(137) 9 seconds ago</td>
</tr>
<tr>
<td></td>
<td>sample-app-static-6-busybox-ce83ce978a7a890a0b1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### View Docker Logs

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](https://docs.docker.com/) 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 [Using the `awslogs` Log Driver](https://docs.aws.amazon.com/AmazonECS/latest/developmentguide/using-awslogs.html) (p. 245).

```bash
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
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

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](https://docs.docker.com/) in the Docker documentation.

```bash
docker inspect dc7240fe892a
```

Output:

```
{
  "AppArmorProfile": ",",
}
```
API failures Error Messages

In some cases, an API call that you have triggered through the Amazon ECS console or the AWS CLI exits with a failures error message. The following possible API failures error messages are explained below for each API call. The failures occur on a particular resource, and the resource in parentheses is the resource associated with the failure.

Many resources are region-specific, so make sure that the console is set to the correct region for your resources. Alternatively, make sure that your AWS CLI commands are being sent to the correct region with the --region region option.

- **DescribeClusters**
  
  MISSING (cluster ID)

  Your cluster was not found. The cluster name may not have been spelled correctly or the wrong region may be specified.

- **DescribeInstances**
  
  MISSING (container instance ID)

  The container instance you are attempting to describe does not exist. Perhaps the wrong cluster or region has been specified, or the container instance ARN or ID is misspelled.

- **DescribeServices**
  
  MISSING (service ID)

  The service you are attempting to describe does not exist. Perhaps the wrong cluster or region has been specified, or the container instance ARN or ID is misspelled.

- **DescribeTasks**
  
  MISSING (task ID)

  The task you are trying to describe does not exist. Perhaps the wrong cluster or region has been specified, or the task ARN or ID is misspelled.

- **RunTask or StartTask**
RESOURCE: * (container instance ID)

The resource or resources requested by the task are unavailable on the given container instance. If the resource is CPU, memory, ports, or elastic network interfaces, you may need to add container instances to your cluster. For RESOURCE: ENI errors, your cluster does not 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 per instance type, see IP Addresses Per Network Interface Per Instance Type in the Amazon EC2 User Guide for Linux Instances.

RESOURCE: GPU

The number of GPUs requested by the task are unavailable on the given container instance. You may need to add GPU-enabled container instances to your cluster. For more information, see Working with GPUs on Amazon ECS (p. 38).

AGENT (container instance ID)

The container instance that you attempted to launch a task onto has an agent that is currently disconnected. To prevent extended wait times for task placement, the request was rejected.

ATTRIBUTE (container instance ID)

Your task definition contains a parameter that requires a specific container instance attribute that is not 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 Task Definition Parameters (p. 195) and Amazon ECS Container Agent Configuration (p. 152).

Note

Windows container instances with Amazon ECS container agent versions earlier than 1.17.0 do not support the awslogs log driver by default. If you are unable to use the awslogs log driver with your Windows container instances, ensure that you are using the latest Amazon ECS-optimized Windows AMI.

• StartTask

MISSING (container instance ID)

The container instance you attempted to launch the task onto does not exist. Perhaps the wrong cluster or region has been specified, or the container instance ARN or ID is misspelled.

INACTIVE (container instance ID)

The container instance that you attempted to launch a task onto was previously deregistered with Amazon ECS and cannot be used.

Troubleshooting IAM Roles for Tasks

If you are having trouble configuring IAM roles for tasks in your cluster, you can try this known good configuration to help debug your own configuration.

The following procedure helps you to:

• Create a CloudWatch Logs log group to store your test logs.
• Create a task IAM role that has full Amazon ECS permissions.
• Register a task definition with a known good AWS CLI configuration that is compatible with IAM roles for tasks.
• Run a task from that task definition to test your container instance support for IAM roles for tasks.
• View the container logs from that task in CloudWatch Logs to verify that it works.

To test IAM roles for tasks with a known good configuration

1. Create a CloudWatch Logs log group called ecs-tasks.
   b. In the left navigation pane, choose Logs, Actions, Create log group.
   c. For Log Group Name, enter ecs-tasks and choose Create log group.
2. Create an IAM role for your task to use.
   b. In the navigation pane, choose Roles, Create role.
   c. For Select type of trusted entity, choose Elastic Container Service.
   d. For Select your use case, choose Elastic Container Service Task, Next: Permissions.
   e. On the Attached permissions policy page, choose AmazonEC2ContainerServiceFullAccess, Next: Review.
   f. On the Review page, for Role name, enter ECS-task-full-access and choose Create role.
3. Register a task definition that uses your new role.
   a. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
   b. In the navigation pane, choose Task Definitions.
   c. On the Task Definitions page, choose Create new Task Definition.
   d. On the Select launch type compatibility page, choose EC2, Next step.
   e. Scroll to the bottom of the page and choose Configure via JSON.
   f. Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose Save.

   **Note**
   Replace the awslogs-region value with the region in which you created your CloudWatch Logs log group.

```json
{
    "taskRoleArn": "ECS-task-full-access",
    "containerDefinitions": [
        {
            "memory": 128,
            "essential": true,
            "name": "amazonlinux",
            "image": "amazonlinux",
            "entryPoint": ["/bin/bash", "-c"],
            "command": ["yum install -y aws-cli; aws ecs list-tasks --region us-west-2"
          ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "ecs-tasks",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "iam-role-test"
                }
            }
        }
    ]
}
```

API Version 2014-11-13
4. Run a task from your task definition.
   a. On the Task Definition: iam-role-test registration confirmation page, choose Actions, Run Task.
   b. On the Run Task page, choose the EC2 launch type, a cluster, and then choose Run Task to run your task.

5. View the container logs in the CloudWatch Logs console.
   b. In the left navigation pane, choose Logs.
   c. Select the ecs-tasks log group.
   d. Select the most recent log stream.
   e. Scroll down to view the last lines of the log stream. You should see the output of the aws ecs list-tasks command.

```json
{
    "taskArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:task/d48feb62-46e2-4cbc-a36b-e0400b993d1d"
    ]
}
```

If you receive an "Unable to locate credentials" error, then the following are possible causes.

- The IAM roles for tasks feature is not enabled on your container instances. For more information, see Enabling Task IAM Roles on your Container Instances (p. 437).
- The credential URL is being throttled. You can use the ECS_TASK_METADATA_RPS_LIMIT container agent parameter to configure the throttle limits. For more information, see Amazon ECS Container Agent Configuration (p. 152).
Windows Containers

Amazon ECS now supports Windows containers on container instances that are launched with the Amazon ECS-optimized Windows AMI.

Windows container instances use their own version of the Amazon ECS container agent. On the Amazon ECS-optimized Windows AMI, the Amazon ECS container agent runs as a service on the host. Unlike the Linux platform, the agent does not run inside a container because it uses the host's registry and the named pipe at `\.\pipe\docker_engine` to communicate with the Docker daemon.

The source code for the Amazon ECS container agent is available on GitHub. We encourage you to submit pull requests for changes that you would like to have included. However, we do not currently provide support for running modified copies of this software. You can view open issues for Amazon ECS and Windows on our GitHub issues page.

Amazon ECS vends AMIs that are optimized for Windows containers in the following variants.

- **Amazon ECS-optimized Windows 2019 AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows 2016 AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.

Topics

- Windows Container Caveats (p. 610)
- Getting Started with Windows Containers (p. 611)
- Windows Task Definitions (p. 616)
- Windows IAM Roles for Tasks (p. 619)
- Pushing Windows Images to Amazon ECR (p. 620)

Windows Container Caveats

Here are some things you should know about Windows containers and Amazon ECS.

- Windows containers cannot run on Linux container instances and vice versa. To ensure proper task placement for Windows and Linux tasks, you should 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 only supported for tasks that use the EC2 launch type. The Fargate launch type is not currently supported for Windows containers. For more information about launch types, see Amazon ECS Launch Types (p. 227).
- Windows containers and container instances cannot support all the task definition parameters that are available for Linux containers and container instances. For some parameters, they are not supported at all, and others behave differently on Windows than they do on Linux. For more information, see Windows Task Definitions (p. 616).
• The IAM roles for tasks feature requires that you configure your Windows container instances to allow the feature at launch, and your containers must run some provided PowerShell code when they use the feature. For more information, see Windows IAM Roles for Tasks (p. 619).
• 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 Service Load Balancing (p. 306).
• The Windows server Docker images are large (9 GiB), so your container instances require more storage space than Linux container instances, which typically have smaller image sizes.

Getting Started with Windows Containers

This tutorial walks you through manually getting Windows containers running on Amazon ECS with the Amazon ECS-optimized Windows AMI. You create a cluster for your Windows container instances, launch one or more container instances into your cluster, register a task definition that uses a Windows container image, create a service that uses that task definition, and then view the sample webpage that the container runs.

Topics
• Step 1: Create a Windows Cluster (p. 611)
• Step 2: Launching a Windows Container Instance into your Cluster (p. 612)
• Step 3: Register a Windows Task Definition (p. 614)
• Step 4: Create a Service with Your Task Definition (p. 615)
• Step 5: View Your Service (p. 616)

Step 1: Create a Windows Cluster

You should create a new cluster for your Windows containers. Linux container instances cannot run Windows containers, and vice versa, so proper task placement is best accomplished by running Windows and Linux container instances in separate clusters. In this tutorial, you create a cluster called windows for your Windows containers.

To create a cluster with the AWS Management Console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Clusters.
3. On the Clusters page, choose Create Cluster.
5. For Cluster name enter a name for your cluster (in this example, windows is the name of the cluster). Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. Choose Create an empty cluster, Create.

To create a cluster with the AWS CLI

• You can create a cluster using the AWS CLI with the following command:

    aws ecs create-cluster --cluster-name windows
Step 2: Launching a Windows Container Instance into your Cluster

You can launch a Windows container instance using the AWS Management Console, as described in this topic. Before you begin, be sure that you’ve completed the steps in Setting Up with Amazon ECS (p. 7). After you’ve launched your instance, you can use it to run tasks.

To launch a Windows container instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the navigation bar, select the region to use.
3. From the console dashboard, choose Launch Instance.
4. On the Choose an Amazon Machine Image (AMI) page, type ECS_Optimized in the Search community AMIs field and press the Enter key. Choose Select next to the Windows_Server-2019-English-Full-ECS_Optimized-2019.05.10 AMI.

The following table lists the current Amazon ECS-optimized Windows 2019 AMI IDs by Region.

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-east-2</td>
<td>ami-0a4548e9bef884a63</td>
</tr>
<tr>
<td>us-east-1</td>
<td>ami-0f7cc2a4e9cb93130</td>
</tr>
<tr>
<td>us-west-1</td>
<td>ami-0ca9f58a76b75d76</td>
</tr>
<tr>
<td>us-west-2</td>
<td>ami-09a6b4fc9786621ef</td>
</tr>
<tr>
<td>ap-northeast-1</td>
<td>ami-0de9f680eb139f5f2</td>
</tr>
<tr>
<td>ap-northeast-2</td>
<td>ami-052dc171cf22efb2c</td>
</tr>
<tr>
<td>ap-south-1</td>
<td>ami-0fda45670ecdda47</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>ami-0d073901cb231d495</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>ami-06cef3b9805e5ebb0</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>ami-09f37f76841876c2b</td>
</tr>
<tr>
<td>cn-north-1</td>
<td>ami-02942c66816678482</td>
</tr>
<tr>
<td>cn-northwest-1</td>
<td>ami-066eff0f2473d2ba3</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>ami-09bff64c8c3102238</td>
</tr>
<tr>
<td>eu-north-1</td>
<td>ami-078d39ec1c8b11d6b</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>ami-05da69b2d804943e6</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>ami-01fbd6d84ec8b36d3</td>
</tr>
<tr>
<td>eu-west-3</td>
<td>ami-0a64405322f93a0c7</td>
</tr>
<tr>
<td>sa-east-1</td>
<td>ami-0ac8048de25ce4284</td>
</tr>
<tr>
<td>us-gov-east-1</td>
<td>ami-0c4c54ea7fe80d45a</td>
</tr>
</tbody>
</table>
**Step 2: Launching a Windows Container Instance into your Cluster**

<table>
<thead>
<tr>
<th>Region</th>
<th>AMI ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>us-gov-west-1</td>
<td>ami-d91f63b8</td>
</tr>
</tbody>
</table>

**Note**
There are Amazon ECS-optimized AMIs for both Windows Server 2019 and Windows Server 2016. For more information, see How to Launch the Latest Amazon ECS-Optimized Amazon Linux AMI (p. 72).

5. On the **Choose an Instance Type** page, you can select the hardware configuration of your instance. The `t2.micro` instance type is selected by default. The instance type that you select determines the resources available for your tasks to run on.

6. Choose **Next: Configure Instance Details**.

7. On the **Configure Instance Details** page, set the **Auto-assign Public IP** check box depending on whether to make your instance accessible from the public internet. If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If your instance should not be accessible from the internet, choose **Disable**.

**Note**
Container instances need external network access to communicate with the Amazon ECS service endpoint, so if 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 HTTP Proxy Configuration (p. 182) in this guide. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters (p. 578).

8. On the **Configure Instance Details** page, select the **ecsInstanceRole** IAM role value that you created for your container instances in Setting Up with Amazon ECS (p. 7).

**Important**
If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent does not connect to your cluster. For more information, see Amazon ECS Container Instance IAM Role (p. 415).

9. Expand the **Advanced Details** section and paste the provided user data PowerShell script into the **User data** field. By default, this script registers your container instance into the `windows` cluster that you created earlier. To launch into another cluster instead of `windows`, replace the red text in the script below with the name of your cluster.

**Note**
The `-EnableTaskIAMRole` option is required to enable IAM roles for tasks. For more information, see Windows IAM Roles for Tasks (p. 619).

```powershell
Import-Module ECSTools
Initialize-ECSAgent -Cluster 'windows' -EnableTaskIAMRole
</powershell>
```

10. Choose **Next: Add Storage**.

11. On the **Add Storage** page, configure the storage for your container instance. The Windows OS and container images are large (approximately 9 GiB for the Windows server core base layers), and just a few images and containers quickly fill up the default 50-GiB volume size for the Amazon ECS-optimized Windows AMI. A larger root volume size (for example, 200 GiB) allows for more containers and images on your instance.

    You can optionally increase or decrease the volume size for your instance to meet your application needs.

12. Choose **Review and Launch**.
13. On the **Review Instance Launch** page, under **Security Groups**, you see that the wizard created and selected a security group for you. By default, you should have port 3389 for RDP connectivity. To have your containers to receive inbound traffic from the internet, open those ports as well.

   a. Choose **Edit security groups**.
   b. On the **Configure Security Group** page, ensure that the **Create a new security group** option is selected.
   c. Add rules for any other ports that your containers may need and choose **Review and Launch**. The sample task definition later in this walkthrough uses port 8080, so you should open that to **Anywhere**.

14. On the **Review Instance Launch** page, choose **Launch**.

15. In the **Select an existing key pair or create a new key pair** dialog box, choose **Choose an existing key pair**, then select the key pair that you created when getting set up.

   When you are ready, select the acknowledgment field, and then choose **Launch Instances**.

16. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.

17. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is **pending**. After the instance starts, its state changes to **running**, and it receives a public DNS name. (If the **Public DNS** column is hidden, choose the **Show/Hide** icon and choose **Public DNS**.)

18. After your instance has launched, you can view your cluster in the Amazon ECS console to see that your container instance has registered with it.

   **Note**
   It can take up to 15 minutes for your Windows container instance to register with your cluster.

### Step 3: 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 **microsoft/iis** container image.

**To register the sample task definition with the AWS Management Console**

1. Open the Amazon ECS console at [https://console.aws.amazon.com/ecs/](https://console.aws.amazon.com/ecs/).
2. In the navigation pane, choose **Task Definitions**.
3. On the **Task Definitions** page, choose **Create new Task Definition**.
4. On the **Select launch type compatibilities** page, choose **EC2**, **Next step**.

   **Note**
   The Fargate launch type is not compatible with Windows containers.

5. Scroll to the bottom of the page and choose **Configure via JSON**.
6. Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose **Save**.

```json
{
    "family": "windows-simple-iis",
    "containerDefinitions": [
        {
            "name": "windows_sample_app",
            "image": "microsoft/iis"
        }
    ]
}
```
"cpu": 512,
"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"],
"portMappings": [
  {
    "protocol": "tcp",
    "containerPort": 80,
    "hostPort": 8080
  }
],
"memory": 768,
"essential": true
]

7. Verify your information and choose Create.

To register the sample task definition with the AWS CLI

1. Create a file called windows-simple-iis.json.
2. Open the file with your favorite text editor and add the sample JSON above to the file and save it.
3. Using the AWS CLI, run the following command to register the task definition with Amazon ECS.
   
   Note
   Make sure that your AWS CLI is configured to use the same region that your Windows cluster exists in, or add the --region your_cluster_region option to your command.

   ```shell
   aws ecs register-task-definition --cli-input-json file://windows-simple-iis.json
   ```

Step 4: 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 on your cluster.

To create a service from your task definition with the console

2. On the Create Service page, enter the following information and then choose Create service.
   - Launch type: EC2
   - Cluster: windows
   - Service name: windows-simple-iis
   - Service type: REPLICA
   - Number of tasks: 1
   - Deployment type: Rolling update

To create a service from your task definition with the AWS CLI

- Using the AWS CLI, run the following command to create your service.
Step 5: 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 Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the **Clusters** page, choose the **windows** cluster.
3. In the **Services** tab, choose the **windows-simple-iis** service.
4. On the **Service: windows-simple-iis** page, choose the task ID for the task in your service.
5. On the **Task** page, expand the **iis** container to view its information.
6. In the **Network bindings** of the container, you should see an **External Link** IP address and port combination link. Choose that link to open the IIS test page in your browser.

### Windows Task Definitions

Windows containers and container instances cannot support all the task definition parameters that are available for Linux containers and container instances. For some parameters, they are not supported at all, and others behave differently on Windows than they do on Linux.

#### Windows Task Definition Parameters

The following list explains which parameters are not supported or behave differently on Windows containers than they do with Linux containers. For more information about these parameters as they relate to Amazon ECS, see [Task Definition Parameters (p. 195)](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-definition-parameters.html).

- **taskRoleArn**

  Supported: Yes
IAM roles for tasks on Windows require that the `-EnableTaskIAMRole` option is set when you launch the Amazon ECS-optimized Windows AMI. Your containers must also run some configuration code in order to take advantage of the feature. For more information, see Windows IAM Roles for Tasks (p. 619).

**networkMode**

Supported: No

Docker for Windows uses different network modes than Docker for Linux. When you register a task definition with Windows containers, you must not specify a network mode. If you use the console to register a task definition with Windows containers, you must choose the `<default>` network mode object.

**containerDefinitions**

Supported: Yes

Additional notes: Not all container definition parameters are supported. Review the list below for individual parameter support.

**portMappings**

Supported: Limited

Port mappings on Windows use the `NetNAT` gateway address rather than `localhost`. There is no loopback for port mappings on Windows, so you cannot access a container's mapped port from the host itself.

**cpu**

Supported: Yes

Amazon ECS treats this parameter in the same manner that it does for Linux containers: if you provide 500 CPU shares to a container, that number of CPU shares is removed from the available resources on the container instance when the task is placed. However, on a Windows container instance, the CPU limit is enforced as an absolute limit, or a quota. Windows containers only have access to the specified amount of CPU that is described in the task definition.

**disableNetworking**

Supported: No

**dnsServers**

Supported: No

**dnsSearchDomains**

Supported: No

**dockerSecurityOptions**

Supported: No

**extraHosts**

Supported: No

**links**

Supported: No

**mountPoints**

Supported: Limited

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives.
Windows Sample Task Definitions

Below is a sample task definition to help you get started with Windows containers on Amazon ECS.

```json

linuxParameters
Supported: No

privileged
Supported: No

readonlyRootFilesystem
Supported: No

user
Supported: No

ulimits
Supported: No

to volumes
Supported: Yes

name
Supported: Yes

dockerVolumeConfiguration
Supported: No

host
Supported: Limited

Windows containers can mount whole directories on the same drive as $env:ProgramData. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives. For example, you can mount C:\my\path:C:\my\path and D:\D:\, but not D:\my\path:C:\my\path or D:\C:\my\path.

cpu
Supported: No

Task-level CPU is ignored for Windows containers. We recommend specifying container-level CPU for Windows containers.

memory
Supported: No

Task-level memory is ignored for Windows containers. We recommend specifying container-level memory for Windows containers.

proxyConfiguration
Supported: No

ipcMode
Supported: No

pidMode
Supported: No

```
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.

```json
{
    "family": "windows-simple-iis",
    "containerDefinitions": [
        {
            "name": "windows_sample_app",
            "image": "microsoft/iis",
            "cpu": 512,
            "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,
                    "hostPort": 8080
                }
            ],
            "memory": 1024,
            "essential": true
        }
    ]
}
```

Windows IAM Roles for Tasks

The IAM roles for tasks with Windows features requires extra configuration, but much of this configuration is similar to enabling IAM roles for tasks on Linux container instances. The following requirements must be met to enable IAM roles for tasks for Windows containers.

- When you launch your container instances, you must enable the feature by setting the `-EnableTaskIAMRole` option in the container instances user data script. 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 IAM Roles for Task Container Bootstrap Script (p. 620).

- You must create an IAM role and policy for your tasks. For more information, see Creating an IAM Role and Policy for your Tasks (p. 437).

- Your container must use an AWS SDK that supports IAM roles for tasks. For more information, see Using a Supported AWS SDK (p. 439).

- You must specify the IAM role you created for your tasks when you register the task definition, or as an override when you run the task. For more information, see Specifying an IAM Role for your Tasks (p. 439).

- The IAM roles for the task credential provider use port 80 on the container instance, so if you enable IAM roles for tasks on your container instance, your containers cannot 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, and the traffic can be
routed to another host port on your container instances. For more information, see Service Load
Balancing (p. 306).

IAM Roles for Task 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.

```bash
# Copyright 2014-2016 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
```

Pushing Windows Images to Amazon ECR

You can push Windows Docker container images to Amazon ECR. You must be using a version of
Docker that supports Windows containers. The following procedures show you how to pull a Windows
Docker image, create an Amazon ECR repository to store the image, tag the image to that repository,
authenticate the image to the Amazon ECR registry, and then push the image to that repository.

To pull and tag a Windows Docker image

1. Pull a Windows Docker image locally. This example uses the microsoft/iis image.

   ```bash
   PS C:\> docker pull microsoft/iis
   Using default tag: latest
   latest: Pulling from microsoft/iis
   3889bb8d880b: Pull complete
   04ee5d718c7a: Pull complete
   c0931dd15237: Pull complete
   61784b745c20: Pull complete
d05122f129ca: Pull complete
   Digest: sha256:25586570b058da9882d4af640d326d0cc26d06b67e1cee63f35ea54d83c882
   Status: Downloaded newer image for microsoft/iis:latest
   ```

2. Create an Amazon ECR repository for your image.

   ```bash
   PS C:\> aws ecr create-repository --repository-name iis
   { "repository": {
   ```
3. Tag the image with the repositoryUri that was returned from the previous command.

   PS C:\> docker tag microsoft/iis 111122223333.dkr.ecr.us-west-2.amazonaws.com/iis

4. Authenticate your Docker client to the Amazon ECR registry.

   **Note**
   The `get-login` command is available in the AWS CLI starting with version 1.9.15; however, we recommend version 1.11.91 or later for recent versions of Docker (17.06 or later). You can check your AWS CLI version with the `aws --version` command. If you are using Docker version 17.06 or later, include the `--no-include-email` option after `get-login`. If you receive an Unknown options: `--no-include-email` error, install the latest version of the AWS CLI. For more information, see `Installing the AWS Command Line Interface` in the AWS Command Line Interface User Guide.

   PS C:\> Invoke-Expression -Command (aws ecr get-login)

5. Push the image to Amazon ECR.

   PS C:\> docker push 111122223333.dkr.ecr.us-west-2.amazonaws.com/iis
   The push refers to a repository [111122223333.dkr.ecr.us-west-2.amazonaws.com/iis]
   1e4f77a75bd4: Pushed
   ac90fbd2a567: Pushed
   c7090349c7b3: Pushed
   b9454c3094c6: Skipped foreign layer
   3fd27e6f6a3: Skipped foreign layer
   latest: digest: sha256:0ddc7af8691072bb22b3f189388b33604c90774d3dc0485b1bf379f9bec4c5
   size: 1574
# 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.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Fargate platform version 1.3.0 update</td>
<td>Beginning on May 1, 2019, any new Fargate task that is launched supports the <code>splunk</code> log driver in addition to the <code>awslogs</code> log driver. For more information, see Storage and Logging (p. 207).</td>
<td>1 May 2019</td>
</tr>
<tr>
<td>AWS Fargate platform version 1.3.0 update</td>
<td>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 <code>secretOptions</code> container definition parameter. For more information, see Specifying Sensitive Data (p. 254).</td>
<td>1 May 2019</td>
</tr>
<tr>
<td>AWS Fargate platform version 1.3.0 update</td>
<td>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 Specifying Sensitive Data (p. 254).</td>
<td>2 Apr 2019</td>
</tr>
<tr>
<td>AWS Fargate platform version 1.3.0 update</td>
<td>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 (p. 225), Container Dependency (p. 216), and Container Timeouts (p. 217).</td>
<td>27 Mar 2019</td>
</tr>
<tr>
<td>Amazon ECS introduces the external deployment type</td>
<td>The <em>external</em> 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 External Deployment (p. 302).</td>
<td>27 Mar 2019</td>
</tr>
<tr>
<td>Getting Started with AWS App Mesh and Amazon ECS</td>
<td>Added tutorial for AWS App Mesh and Amazon ECS. For more information, see Getting Started with AWS App Mesh and Amazon ECS (p. 570).</td>
<td>27 Mar 2019</td>
</tr>
<tr>
<td>AWS Deep Learning Containers on Amazon ECS</td>
<td>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 AWS Deep Learning Containers on Amazon ECS (p. 577).</td>
<td>27 Mar 2019</td>
</tr>
<tr>
<td>Amazon ECS introduces enhanced container dependency management</td>
<td>Amazon ECS introduces additional task definition parameters that enable you to define dependencies for container startup and shutdown as well as a per-</td>
<td>7 Mar 2019</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Change</td>
<td>container start and stop timeout value. For more information, see Container Dependency (p. 216).</td>
<td></td>
</tr>
<tr>
<td>Amazon ECS CLI v1.13.0</td>
<td>New version of the Amazon ECS CLI released. For more information, see Amazon ECS CLI Changelog.</td>
<td>7 Mar 2019</td>
</tr>
<tr>
<td>Amazon ECS introduces the PutAccountSettingDefault API</td>
<td>Amazon ECS introduces the PutAccountSettingDefault API that allows a user to set the default ARN/ID format opt in status for all the IAM users and roles on the account. Previously, setting the account's default opt in status required the use of the root user. For more information, see Amazon Resource Names (ARNs) and IDs (p. 355).</td>
<td>8 Feb 2019</td>
</tr>
<tr>
<td>Amazon ECS supports GPU workloads</td>
<td>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 Working with GPUs on Amazon ECS (p. 38).</td>
<td>4 Feb 2019</td>
</tr>
<tr>
<td>Amazon ECS expanded secrets support</td>
<td>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 Specifying Sensitive Data (p. 254).</td>
<td>21 Jan 2019</td>
</tr>
<tr>
<td>Interface VPC Endpoints (AWS PrivateLink)</td>
<td>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).</td>
<td>26 Dec 2018</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| 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 Specifying Sensitive Data (p. 254).  
  • 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 Fargate Task Recycling (p. 288).  
For more information, see AWS Fargate Platform Versions (p. 36). | 17 Dec 2018 |
| 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 Limits (p. 587). | 14 Dec 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 AWS Fargate Platform Versions (p. 36). | 07 Dec 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 Blue/Green Deployment with CodeDeploy (p. 299). | 27 Nov 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 AMIs (p. 62). | 26 Nov 2018 |
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>New version of the Amazon ECS CLI released, which added the following functionality:</td>
<td>19 Nov 2018</td>
</tr>
<tr>
<td></td>
<td>• Added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers. For more information, see Using Amazon ECS Parameters (p. 546).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Added support for specifying the <code>ipcMode</code> and <code>pidMode</code> Docker flags in task definitions. For more information, see Using Amazon ECS Parameters (p. 546).</td>
<td></td>
</tr>
<tr>
<td>Added support for additional Docker flags in task definitions</td>
<td>Amazon ECS introduced support for the following Docker flags in task definitions:</td>
<td>16 Nov 2018</td>
</tr>
<tr>
<td></td>
<td>• IPC Mode (p. 224)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• PID Mode (p. 225)</td>
<td></td>
</tr>
<tr>
<td>Amazon ECS secrets support</td>
<td>Amazon ECS added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers.</td>
<td>15 Nov 2018</td>
</tr>
<tr>
<td></td>
<td>For more information, see Specifying Sensitive Data (p. 254).</td>
<td></td>
</tr>
<tr>
<td>Resource tagging</td>
<td>Amazon ECS added support for adding metadata tags to your services, task definitions, tasks, clusters, and container instances.</td>
<td>15 Nov 2018</td>
</tr>
<tr>
<td></td>
<td>For more information, see Resources and Tags (p. 355).</td>
<td></td>
</tr>
<tr>
<td>AWS Fargate Region expansion</td>
<td>AWS Fargate with Amazon ECS has expanded to the US West (N. California) and Asia Pacific (Seoul) Regions.</td>
<td>07 Nov 2018</td>
</tr>
<tr>
<td></td>
<td>For more information, see AWS Fargate on Amazon ECS (p. 30).</td>
<td></td>
</tr>
<tr>
<td>Service limits updated</td>
<td>The following service limits were updated:</td>
<td>31 Oct 2018</td>
</tr>
<tr>
<td></td>
<td>• Number of tasks using the Fargate launch type, per Region, per account was raised from 20 to 50.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of public IP addresses for tasks using the Fargate launch type was raised from 20 to 50.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information, see Amazon ECS Service Limits (p. 587).</td>
<td></td>
</tr>
<tr>
<td>AWS Fargate Region expansion</td>
<td>AWS Fargate with Amazon ECS has expanded to the EU (London) Region.</td>
<td>26 Oct 2018</td>
</tr>
<tr>
<td></td>
<td>For more information, see AWS Fargate on Amazon ECS (p. 30).</td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td><strong>Amazon ECS CLI v1.10.0</strong></td>
<td>New version of the Amazon ECS CLI released, which added the following functionality:</td>
<td>25 October 2018</td>
</tr>
<tr>
<td></td>
<td>• Added the <code>ecs-cli registry-creds</code> command, which facilitates the creation and use of private registry credentials within Amazon ECS. For more information, see <code>ecs-cli registry-creds</code> (p. 539).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Added support for Amazon Linux 2. For more information, see <a href="#">Amazon ECS-optimized AMIs</a> (p. 62).</td>
<td></td>
</tr>
<tr>
<td><strong>Amazon ECS-optimized Amazon Linux 2 AMI Released</strong></td>
<td>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 Amazon Linux AMI, 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 <a href="#">Amazon ECS-optimized AMIs</a> (p. 62).</td>
<td>18 October 2018</td>
</tr>
<tr>
<td><strong>Amazon ECS CLI v1.9.0</strong></td>
<td>New version of the Amazon ECS CLI released, which added the following functionality:</td>
<td>18 October 2018</td>
</tr>
<tr>
<td></td>
<td>• Added support for service discovery. For more information, see <a href="#">Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI</a> (p. 468).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Added support for Amazon EC2 Spot instances in Amazon ECS clusters.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Added support for custom user data.</td>
<td></td>
</tr>
<tr>
<td><strong>Amazon ECS Task Metadata Endpoint version 3</strong></td>
<td>Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called <code>ECS_CONTAINER_METADATA_URI</code> 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 <code>awsvpc</code> network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see <a href="#">Amazon ECS Task Metadata Endpoint</a> (p. 172).</td>
<td>18 October 2018</td>
</tr>
<tr>
<td><strong>Amazon ECS service discovery Region expansion</strong></td>
<td>Amazon ECS service discovery has expanded support to the Canada (Central), South America (São Paulo), Asia Pacific (Seoul), Asia Pacific (Mumbai), and EU (Paris) Regions. For more information, see <a href="#">Service Discovery</a> (p. 328).</td>
<td>27 September 2018</td>
</tr>
<tr>
<td><strong>Added support for additional Docker flags in container definitions</strong></td>
<td>Amazon ECS introduced support for the following Docker flags in container definitions:</td>
<td>17 Sept 2018</td>
</tr>
<tr>
<td></td>
<td>• System Controls (p. 218)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Interactive (p. 219)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Pseudo Terminal (p. 219)</td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Private registry authentication support for Amazon ECS using AWS Fargate tasks</td>
<td>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 Private Registry Authentication for Tasks (p. 251).</td>
<td>10 Sept 2018</td>
</tr>
</tbody>
</table>
| Amazon ECS CLI v1.8.0                               | New version of the Amazon ECS CLI released, which added the following functionality:  
• Added support for Docker volumes in Docker compose files. For more information, see ecs-cli compose (p. 500).  
• Added support for task placement constraints and strategies in Docker compose files. For more information, see ecs-cli compose (p. 500).  
• Added support for private registry authentication in Docker compose files. For more information, see ecs-cli compose (p. 500).  
• Added support for --force-update on compose up to force relaunching of tasks. For more information, see ecs-cli compose up (p. 509). | 7 Sept 2018    |
<p>| 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 EU (London) Regions. For more information, see Service Discovery (p. 328). | 30 August 2018|
| Scheduled tasks with Fargate tasks support          | Amazon ECS introduced support for scheduled tasks for the Fargate launch type. For more information, see Scheduled Tasks (cron) (p. 283). | 28 August 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 Private Registry Authentication for Tasks (p. 251). | 16 August 2018|
| Docker volume support added                         | Amazon ECS introduced support for Docker volumes. For more information, see Using Data Volumes in Tasks (p. 229). | 9 August 2018  |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Fargate Region expansion</td>
<td>AWS Fargate with Amazon ECS has expanded to the EU (Frankfurt), Asia Pacific (Singapore), and Asia Pacific (Sydney) Regions. For more information, see AWS Fargate on Amazon ECS (p. 30).</td>
<td>19 July 2018</td>
</tr>
<tr>
<td>Amazon ECS CLI v1.7.0</td>
<td>New version of the Amazon ECS CLI released, which added the following functionality:</td>
<td>18 July 2018</td>
</tr>
<tr>
<td></td>
<td>• Added support for container healthcheck and devices in Docker compose files. For more information, see ecs-cli compose (p. 500).</td>
<td></td>
</tr>
<tr>
<td>Amazon ECS service scheduler strategies added</td>
<td>Amazon ECS introduced the concept of service scheduler strategies. There are two service scheduler strategies available:</td>
<td>12 June 2018</td>
</tr>
<tr>
<td></td>
<td>• 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 (p. 291).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 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 (p. 291).</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fargate tasks do not support the DAEMON scheduling strategy.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information, see Service Scheduler Concepts (p. 290).</td>
<td></td>
</tr>
<tr>
<td>Amazon ECS CLI v1.6.0</td>
<td>New version of the Amazon ECS CLI released, which added the following functionality:</td>
<td>5 June 2018</td>
</tr>
<tr>
<td></td>
<td>• Added support for Docker compose file syntax version 3. For more information, see ecs-cli compose (p. 500).</td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
</tbody>
</table>
| **Amazon ECS container agent v1.18.0** | New version of the Amazon ECS container agent released, which added the following functionality:  
  • Added procedure to manually install the container agent from a S3 URL on non-Amazon Linux EC2 instance, including a PGP signature method for verifying the Amazon ECS container agent installation file. For more information, see Installing the Amazon ECS Container Agent (p. 133).  
  • Added procedure to manually install the container agent from a S3 URL on a Windows EC2 instance, including a PGP signature method for verifying the Amazon ECS container agent installation file. For more information, see Getting Started with Windows Containers (p. 611).  
  • 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 (p. 152).  
For more information, see [amazon-ecs-agent github](#). | 24 May 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 Service Discovery (p. 328). | 22 May 2018 |
| **Added support for additional Amazon ECS-optimized AMI metadata parameters** | Added subparameters that allow you to programmatically retrieve the Amazon ECS-optimized AMI ID, image name, operating system, container agent version, and runtime version. Query the metadata using the SSM Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 91). | 9 May 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 on Amazon ECS (p. 30). | 26 April 2018 |
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
</table>
| Amazon ECS CLI v1.5.0 | New version of the Amazon ECS CLI released, which added the following functionality:  
  - Added support for the ECS CLI to automatically retrieve the latest stable Amazon ECS-optimized AMI by querying the SSM Parameter Store API during the cluster resource creation process. This requires the user account that you are using to have the required SSM permissions. For more information, see `ecs-cli up` (p. 479).  
  - Added support for the `shm_size` and `tmpfs` parameters in compose files. For more information, see `ecs-cli compose` (p. 500).  
  For more information about the updated ECS CLI syntax, see Amazon ECS Command Line Reference (p. 471). | 19 April 2018 |
| Amazon ECS-optimized AMI Metadata Retrieval | Added ability to programatically retrieve Amazon ECS-optimized AMI metadata using the SSM Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 91). | 10 April 2018 |
| Amazon ECS CLI download verification | Added new PGP signature method for verifying the Amazon ECS CLI installation file. For more information, see Installing the Amazon ECS CLI (p. 450). | 5 April 2018 |
| AWS Fargate Platform Version | New AWS Fargate platform version released, which contains:  
  - Added support for Amazon ECS Task Metadata Endpoint (p. 172).  
  - Added support for Health Check (p. 201).  
  - Added support for Service Discovery (p. 328)  
  For more information, see AWS Fargate Platform Versions (p. 36). | 26 March 2018 |
| Amazon ECS Service Discovery | Added integration with Route 53 to support Amazon ECS service discovery. For more information, see Service Discovery (p. 328). | 22 March 2018 |
| Amazon ECS CLI v1.4.2 | New version of the Amazon ECS CLI released, which added the following functionality:  
  - Updated the AMI to `amzn-ami-2017.09.k-amazon-ecs-optimized`.  
  For more information about the updated ECS CLI syntax, see Amazon ECS Command Line Reference (p. 471). | 20 March 2018 |
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker shm-size and tmpfs support</td>
<td>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 (p. 214).</td>
<td>20 March 2018</td>
</tr>
</tbody>
</table>
| Amazon ECS CLI v1.4.0 | New version of the Amazon ECS CLI released, which added the following functionality:  
- Added support for the us-gov-west-1 Region.  
- Added `--force-deployment` flag for the compose service command. For more information, see `ecs-cli compose service` (p. 511).  
- Added support for `aws_session_token` in ECS profiles. For more information, see `ecs-cli configure profile` (p. 476).  
- Updated the AMI to `amzn-ami-2017.09.j-amazon-ecs-optimized`.  
For more information about the updated ECS CLI syntax, see Amazon ECS Command Line Reference (p. 471). | 09 March 2018 |
<p>| Container Health Checks | Added support for Docker health checks in container definitions. For more information, see Health Check (p. 201). | 08 March 2018 |
| AWS Fargate | Added overview for Amazon ECS with AWS Fargate. For more information, see AWS Fargate on Amazon ECS (p. 30). | 22 February 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 <code>awsvpc</code> network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see Amazon ECS Task Metadata Endpoint (p. 172). | 8 February 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 Target Tracking Scaling Policies (p. 321). Removed the previous tutorial for step scaling in the ECS first run wizard. This was replaced with the new tutorial for target tracking. | 8 February 2018 |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon ECS CLI v1.3.0</td>
<td>New version of the Amazon ECS CLI released, which added the following functionality:</td>
<td>19 January 2018</td>
</tr>
<tr>
<td></td>
<td>• Ability to create empty clusters with the up command.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Added --health-check-grace-period flag for the compose service up command.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Updated the AMI to amzn-ami-2017.09.g-amazon-ecs-optimized.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information about the updated ECS CLI syntax, see Amazon ECS Command Line Reference (p. 471).</td>
<td></td>
</tr>
<tr>
<td>Docker 17.09 support</td>
<td>Added support for Docker 17.09. For more information, see Amazon ECS-optimized AMIs (p. 62).</td>
<td>18 January 2018</td>
</tr>
<tr>
<td>Elastic Load Balancing health check initialization wait period</td>
<td>Added ability to specify a wait period for health checks.</td>
<td>27 December 2017</td>
</tr>
<tr>
<td>New service scheduler behavior</td>
<td>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. For more information about this updated behavior, see Additional Service Concepts (p. 292).</td>
<td>11 January 2018</td>
</tr>
<tr>
<td>Task-level CPU and memory</td>
<td>Added support for specifying CPU and memory at the task-level in task definitions. For more information, see TaskDefinition.</td>
<td>12 December 2017</td>
</tr>
<tr>
<td>Amazon ECS console CodePipeline integration</td>
<td>Added Amazon ECS integration with CodePipeline. CodePipeline supports Amazon ECS as a deployment option to help set up deployment pipelines. For more information, see Tutorial: Continuous Deployment with CodePipeline (p. 49).</td>
<td>12 December 2017</td>
</tr>
<tr>
<td>Task execution role</td>
<td>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:</td>
<td>7 December 2017</td>
</tr>
<tr>
<td></td>
<td>• Calls to Amazon ECR to pull the container image</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Calls to CloudWatch to store container application logs</td>
<td></td>
</tr>
<tr>
<td>Windows containers support GA</td>
<td>Added support for Windows 2016 containers. For more information, see Windows Containers (p. 610).</td>
<td>5 December 2017</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| Amazon ECS CLI v1.1.0 with Fargate support | New version of the Amazon ECS CLI released, which added the following features:  
  - Support for task networking  
  - Support for AWS Fargate  
  - Support for viewing CloudWatch Logs data from a task  
  For more information, see [ECS CLI changelog](https://docs.aws.amazon.com/ecs/latest/containerinstances/ecs-cli-changelog.html). | 29 November 2017 |
| AWS Fargate GA | Added support for launching Amazon ECS services using the Fargate launch type. For more information, see [Amazon ECS Launch Types](https://docs.aws.amazon.com/ecs/latest/userguide/task-definition-launch-types.html). | 29 November 2017 |
| Amazon ECS name change | Amazon Elastic Container Service is renamed (previously Amazon EC2 Container Service). | 21 November 2017 |
| 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 [Task Networking with the awsvpc Network Mode](https://docs.aws.amazon.com/ecs/latest/containerinstances/task-networking.html). | 14 November 2017 |
| Amazon ECS CLI v1.0.0 | New version of the Amazon ECS CLI released, which added the following features:  
  - Support for adding multiple named profiles and cluster configurations  
  - Support for custom task definition parameters specified using `--ecs-params`  
  - Support for running the Amazon ECS CLI on Windows  
  For more information, see [ECS CLI changelog](https://docs.aws.amazon.com/ecs/latest/containerinstances/ecs-cli-changelog.html). | 7 November 2017 |
<p>| 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 <a href="https://docs.aws.amazon.com/ecs/latest/containerinstances/container-metadata-file.html">Amazon ECS Container Metadata File</a>. | 2 November 2017 |
| Docker 17.06 support | Added support for Docker 17.06. For more information, see <a href="https://aws.amazon.com/ecs/optimized-amis/">Amazon ECS-optimized AMIs</a>. | 2 November 2017 |
| Support for Docker flags: device and init | Added support for Docker's device and init features in task definitions using the <code>LinuxParameters</code> parameter (device and <code>initProcessEnabled</code>). For more information, see <a href="https://docs.aws.amazon.com/ecs/latest/containerinstances/container-definition.html#linuxparameters">LinuxParameters</a>. | 2 November 2017 |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for Docker flags: cap-add and cap-drop</td>
<td>Added support for Docker's cap-add and cap-drop features in task definitions using the <code>LinuxParameters</code> parameter (capabilities). For more information, see <code>LinuxParameters</code>.</td>
<td>22 September 2017</td>
</tr>
<tr>
<td>Network Load Balancer support</td>
<td>Amazon ECS added support for Network Load Balancers in the Amazon ECS console. For more information, see <code>Creating a Network Load Balancer (p. 316)</code>.</td>
<td>7 September 2017</td>
</tr>
<tr>
<td>RunTask overrides</td>
<td>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 <code>Running Tasks (p. 269)</code>.</td>
<td>27 June 2017</td>
</tr>
<tr>
<td>Amazon ECS scheduled tasks</td>
<td>Added support for scheduling tasks using cron. For more information, see <code>Scheduled Tasks (cron) (p. 283)</code>.</td>
<td>7 June 2017</td>
</tr>
<tr>
<td>Spot Instances in the Amazon ECS console</td>
<td>Added support for creating Spot Fleet container instances within the Amazon ECS console. For more information, see <code>Launching an Amazon ECS Container Instance (p. 101)</code>.</td>
<td>6 June 2017</td>
</tr>
</tbody>
</table>
| Amazon ECS CLI v0.5.0 | New version of the Amazon ECS CLI released, which added the following features:  
  - Ability to push, pull, and list Amazon ECR images  
  - Support for existing load balancers and Application Load Balancers in CreateService  

For more information, see `ECS CLI changelog`. | 3 April 2017 |
<p>| Amazon SNS notification for new Amazon ECS-optimized AMI releases | Added ability to subscribe to SNS notifications about new Amazon ECS-optimized AMI releases. For more information, see <code>Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications (p. 97)</code>. | 23 March 2017 |
| Microservices and batch jobs | Added documentation for two common use cases for Amazon ECS: microservices and batch jobs. For more information, see <code>Common Use Cases in Amazon ECS (p. 567)</code>. | 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 <code>Container Instance Draining (p. 122)</code>. | 24 January 2017 |
| Docker 1.12 support | Added support for Docker 1.12. For more information, see <code>Amazon ECS-optimized AMIs (p. 62)</code>. | 24 January 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 information, see <code>Amazon ECS Task Placement Strategies (p. 274)</code>. | 29 December 2016 |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows container support in beta</td>
<td>Added support for Windows 2016 containers (beta). For more information, see Windows Containers (p. 610).</td>
<td>20 December 2016</td>
</tr>
<tr>
<td>Blox OSS support</td>
<td>Added support for Blox OSS, which allows for custom task schedulers. For more information, see Scheduling Amazon ECS Tasks (p. 268).</td>
<td>1 December 2016</td>
</tr>
<tr>
<td>Amazon ECS event stream for CloudWatch Events</td>
<td>Amazon ECS now sends container instance and task state changes to CloudWatch Events. For more information, see Amazon ECS Event Stream for CloudWatch Events (p. 384).</td>
<td>21 November 2016</td>
</tr>
<tr>
<td>Amazon ECS container logging to CloudWatch Logs</td>
<td>Added support for the awslogs driver to send container log streams to CloudWatch Logs. For more information, see Using the awslogs Log Driver (p. 245).</td>
<td>12 September 2016</td>
</tr>
<tr>
<td>Amazon ECS services with Elastic Load Balancing support for dynamic ports</td>
<td>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 Service Load Balancing (p. 306).</td>
<td>11 August 2016</td>
</tr>
<tr>
<td>IAM roles for Amazon ECS tasks</td>
<td>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 IAM Roles for Tasks (p. 435).</td>
<td>13 July 2016</td>
</tr>
<tr>
<td>Amazon ECS CLI support for Docker Compose v2 format</td>
<td>The Amazon ECS CLI added support for Docker Compose v2 format. For more information, see ecs-cli compose (p. 500).</td>
<td>8 July 2016</td>
</tr>
<tr>
<td>Docker 1.11 support</td>
<td>Added support for Docker 1.11. For more information, see Amazon ECS-optimized AMIs (p. 62).</td>
<td>31 May 2016</td>
</tr>
<tr>
<td>Task automatic scaling</td>
<td>Amazon ECS added support for automatically scaling your tasks run by a service. For more information, see Service Auto Scaling (p. 321).</td>
<td>18 May 2016</td>
</tr>
<tr>
<td>Task definition filtering on task family</td>
<td>Added support for filtering a list of task definition based on the task definition family. For more information, see ListTaskDefinitions.</td>
<td>17 May 2016</td>
</tr>
<tr>
<td>Docker container and Amazon ECS agent logging</td>
<td>Amazon ECS added ability to send ECS agent and Docker container logs from container instances to CloudWatch Logs to simplify troubleshooting issues.</td>
<td>5 May 2016</td>
</tr>
<tr>
<td>Amazon ECS CLI v0.3 released</td>
<td>New version of the Amazon ECS CLI released, which added support for service creation with a load balancer.</td>
<td>11 April 2016</td>
</tr>
<tr>
<td>ECS-optimized AMI now supports Amazon Linux 2016.03.</td>
<td>The ECS-optimized AMI added support for Amazon Linux 2016.03. For more information, see Amazon ECS-optimized AMIs (p. 62).</td>
<td>5 April 2016</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Docker 1.9 support</td>
<td>Added support for Docker 1.9. For more information, see Amazon ECS-optimized AMIs (p. 62).</td>
<td>22 December 2015</td>
</tr>
<tr>
<td>CloudWatch metrics for cluster CPU and memory reservation</td>
<td>Amazon ECS added custom CloudWatch metrics for CPU and memory reservation.</td>
<td>22 December 2015</td>
</tr>
<tr>
<td>Amazon ECR</td>
<td>Added the new Amazon ECR service to the console, which added support for storing images that are controlled by resource-level permissions associated with Docker Hub or IAM users. Available in all AWS Regions, images are automatically replicated and cached globally so that starting hundreds of containers is as fast as a single container.</td>
<td>21 December 2015</td>
</tr>
<tr>
<td>New Amazon ECS first-run experience</td>
<td>The Amazon ECS console first-run experience added zero-click role creation.</td>
<td>23 November 2015</td>
</tr>
<tr>
<td>Task placement across Availability Zones</td>
<td>The Amazon ECS service scheduler added support for task placement across Availability Zones.</td>
<td>8 October 2015</td>
</tr>
<tr>
<td>Amazon ECS CLI with support for Docker Compose</td>
<td>The Amazon ECS CLI added support for Docker Compose.</td>
<td>8 October 2015</td>
</tr>
<tr>
<td>CloudWatch metrics for Amazon ECS clusters and services</td>
<td>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.</td>
<td>17 August 2015</td>
</tr>
<tr>
<td>UDP port support</td>
<td>Added support for UDP ports in task definitions.</td>
<td>7 July 2015</td>
</tr>
<tr>
<td>Environment variable overrides</td>
<td>Added support for deregisterTaskDefinition and environment variable overrides for runTask.</td>
<td>18 June 2015</td>
</tr>
<tr>
<td>Automated Amazon ECS agent updates</td>
<td>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.</td>
<td>11 June 2015</td>
</tr>
<tr>
<td>Amazon ECS service scheduler and Elastic Load Balancing integration</td>
<td>Added ability to define a service and associate that service with an Elastic Load Balancing load balancer.</td>
<td>9 April 2015</td>
</tr>
<tr>
<td>Amazon ECS GA</td>
<td>Amazon ECS general availability in the US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), and EU (Ireland) Regions.</td>
<td>9 April 2015</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.