Amazon Elastic Container Service

Developer Guide

API Version 2014-11-13

Amazon Elastic Container Service: Developer Guide

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

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 a Key Pair ... 9
- Create a Virtual Private Cloud ... 11
- Create a Security Group ... 12
- Install the AWS CLI ... 13

Docker Basics ... 14
- Installing Docker on Amazon Linux 2 ... 14
- Create a Docker Image ... 15
- Push your image to Amazon Elastic Container Registry ... 16
- Clean up ... 17

Repositories ... 18
- Using Amazon ECR Images with Amazon ECS ... 18

Getting Started with Amazon ECS Using Amazon EC2 ... 19
- Prerequisites ... 19
- Step 1: Register a task definition ... 19
- Step 2: Create a cluster ... 20
- Step 3: Create a Service ... 21
- Step 4: View your Service ... 21
- Step 5: Clean Up ... 22

Getting Started with Amazon ECS using Fargate ... 23
- Prerequisites ... 23
- Step 1: Create a Task Definition ... 23
- Step 2: Configure the Service ... 24
- Step 3: Configure the Cluster ... 25
- Step 4: Review ... 25
- Step 5: View your Service ... 25
- Step 6: Clean Up ... 26

AWS Fargate ... 27
- Task Definitions ... 28
  - Network Mode ... 29
  - Task CPU and Memory ... 29
  - Task Resource Limits ... 29
  - Logging ... 29
  - Amazon ECS Task Execution IAM Role ... 30
  - Example Task Definition ... 30
  - Task Storage ... 31
- Tasks and Services ... 31
  - Task Networking ... 32
- Private Registry Authentication ... 32
- Clusters ... 33
- Fargate Spot ... 33
- Fargate Task Retirement ... 33

Fargate Savings Plans ... 33
Platform Versions ... 33
Platform version considerations ... 34
Available platform versions ... 34
Migrating from platform version 1.3.0 to 1.4.0 ... 36
Clusters ... 37
Cluster Concepts ... 37
Creating a Cluster ... 38
Cluster Capacity Providers ... 40
Cluster Capacity Provider Concepts ... 41
Cluster Capacity Provider Considerations ... 41
Cluster Auto Scaling ... 42
Cluster Auto Scaling Considerations ... 42
Auto Scaling group Capacity Providers ... 42
Tutorial: Using Cluster Auto Scaling with the AWS Management Console ... 46
Tutorial: Using Cluster Auto Scaling with the AWS CLI ... 52
Using AWS Fargate Capacity Providers ... 67
Fargate Capacity Provider Considerations ... 67
Handling Fargate Spot Termination Notices ... 68
Creating a New Cluster That Uses Fargate Capacity Providers ... 69
Adding Fargate Capacity Providers To An Existing Cluster ... 69
Running Tasks Using a Fargate Capacity Provider ... 70
Updating Cluster Settings ... 71
Deleting a Cluster ... 71
Task Definitions ... 73
Application Architecture ... 73
Using the Fargate Launch Type ... 74
Using the EC2 Launch Type ... 74
Creating a Task Definition ... 75
Task Definition Template ... 80
Task definition parameters ... 84
Family ... 84
Task role ... 84
Task execution role ... 84
Network mode ... 85
Container Definitions ... 85
Volumes ... 110
Task placement constraints ... 113
Launch types ... 114
Task size ... 114
Proxy configuration ... 116
Other task definition parameters ... 117
Launch Types ... 119
Fargate Launch Type ... 119
EC2 Launch Type ... 120
Working with GPUs on Amazon ECS ... 121
Considerations for Working with GPUs ... 122
Specifying GPUs in Your Task Definition ... 123
Using Data Volumes in Tasks ... 124
Fargate Task Storage ... 125
Docker Volumes ... 126
Bind Mounts ... 131
Amazon EFS Volumes ... 138
Task Networking ... 141
Task Networking Considerations ... 141
Enabling Task Networking ... 143
Using the awslogs Log Driver ... 143

Enabling the awslogs Log Driver for Your Containers ... 144
Creating a Log Group ... 144
Available awslogs Log Driver Options ... 145
Specifying a Log Configuration in your Task Definition ... 146
Viewing awslogs Container Logs in CloudWatch Logs ... 148
Custom Log Routing ... 150
Considerations ... 150
Required IAM Permissions ... 151
Using the AWS for Fluent Bit Image ... 152
Creating a Task Definition that Uses a FireLens Configuration ... 154
Using Fluent Logger Libraries ... 157
Filtering Logs Using Regular Expressions ... 157
Example Task Definitions ... 158
Private Registry Authentication for Tasks ... 160
Required IAM Permissions for Private Registry Authentication ... 161
Enabling Private Registry Authentication ... 162
Specifying Sensitive Data ... 163
Using Secrets Manager ... 163
Using Systems Manager Parameter Store ... 170
Specifying environment variables ... 175
Considerations for specifying environment variable files ... 176
Required IAM permissions ... 176
Example Task Definitions ... 177
Example: Webserver ... 177
Example: `splunk` Log Driver ... 178
Example: `fluentd` Log Driver ... 178
Example: `gelf` Log Driver ... 179
Example: Amazon ECR Image and Task Definition IAM Role ... 179
Example: Entrypoint with Command ... 180
Example: Container Dependency ... 180
Updating a Task Definition ... 181
Deregistering Task Definition Revisions ... 182
Account Settings ... 183
Amazon Resource Names (ARNs) and IDs ... 184
Viewing Account Settings ... 185
Modifying Account Settings ... 186
Container Instances ... 188
Container Instance Concepts ... 188
Container Instance Lifecycle ... 189
Check the Instance Role for Your Account ... 190
Amazon ECS-optimized AMIs ... 190
AMI Versions ... 198
AMI Storage Configuration ... 207
Retrieving Amazon ECS-Optimized AMI Metadata ... 211
Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications ... 214
Amazon SNS Message Format ... 217
Launching a Container Instance ... 219
Using Spot Instances ... 222
Spot Instance Draining ... 222
Bootstrap Container Instances ... 223
Amazon ECS Container Agent ... 223
Docker Daemon ... 224
cloud-init-per Utility ... 224
Specifying Multiple User Data Blocks Using a MIME Multi Part Archive ... 225
Example User Data Scripts ... 226
Elastic Network Interface Trunking ... 230
ENI Trunking Considerations ... 231

Working With Container Instances With Increased ENI Limits ... 231
Supported Amazon EC2 Instance Types ... 234
Connect to Your Container Instance ... 236
CloudWatch Logs ... 237
CloudWatch Logs IAM Policy ... 238
Installing and Configuring the CloudWatch Agent ... 238
Viewing CloudWatch Logs ... 239
Container Instance Draining ... 239
Draining Instances ... 240
Memory Management ... 240
Reserving System Memory ... 241
Viewing Container Instance Memory ... 241
Managing Container Swap Space ... 242
Container Swap Considerations ... 243
Manage Container Instances Remotely ... 243
Run Command IAM Policy ... 244
Installing SSM Agent on an Amazon ECS-Optimized AMI ... 244
Using Run Command ... 245
Starting a Task at Container Instance Launch Time ... 246
Deregister Container Instances ... 248
Container Agent ... 250
Installing the Amazon ECS Container Agent ... 250
Installing the Amazon ECS Container Agent on an Amazon Linux 2 EC2 Instance ... 251
Installing the Amazon ECS Container Agent on an Amazon Linux AMI EC2 Instance ... 251
Installing the Amazon ECS Container Agent on a non-Amazon Linux EC2 Instance ... 252
Running the Amazon ECS Container Agent with Host Network Mode ... 259
Container Agent Versions ... 259
Amazon ECS-Optimized Amazon Linux 2 AMI Container Agent Versions ... 259
Amazon ECS-Optimized Amazon Linux AMI Container Agent Versions ... 261
Updating the Amazon ECS Container Agent ... 264
Checking Your Amazon ECS Container Agent Version ... 265
Updating the Amazon ECS Container Agent on an Amazon ECS-optimized AMI ... 266
Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-Optimized AMIs) ... 268
Amazon ECS Container Agent Configuration ... 270
Available Parameters ... 271
Storing Container Instance Configuration in Amazon S3 ... 283
Private Registry Authentication for Container Instances ... 284
Authentication Formats ... 284
Enabling Private Registries ... 286
Automated Task and Image Cleanup ... 287
Tunable Parameters ... 287
Cleanup Workflow ... 288
Container Metadata File ... 288
Enabling Container Metadata ... 288
Container Metadata File Locations ... 289
Container Metadata File Format ... 289
Task Metadata Endpoint ... 292
Task Metadata Endpoint version 4 ... 292
Task Metadata Endpoint version 3 ... 300
Task Metadata Endpoint version 2 ... 305
Amazon ECS Container Agent Introspection ... 309
HTTP Proxy Configuration ... 311
Amazon Linux Container Instance Configuration ... 311
Windows Container Instance Configuration ... 314
Scheduling Tasks ... 315
Running Tasks ... 316
Running a Task Using the Fargate Launch Type ... 316

Running a Task Using the EC2 Launch Type ... 318
Task Placement ... 320
Task Placement Strategies ... 321
Task Placement Constraints ... 323
Cluster Query Language ... 327
Scheduled Tasks (`cron`) ... 330
Task Lifecycle ... 332
Lifecycle States ... 333
Task Retirement ... 334
Working with Tasks Scheduled for Retirement ... 335
Fargate Task Recycling ... 336
Creating a Scheduled Task Using the AWS CLI ... 336
Services ... 338
Service scheduler concepts ... 338
Daemon ... 339
Replica ... 339
Additional service concepts ... 340
Service Definition Parameters ... 340
Launch Type ... 340
Capacity Provider Strategy ... 341
Task Definition ... 342
Platform Version ... 342
Cluster ... 342
Service Name ... 342
Scheduling Strategy ... 343
Desired Count ... 343
Deployment Configuration ... 343
Deployment Controller ... 345
Task Placement ... 346
Tags ... 347
Network Configuration ... 348
Client Token ... 351
Service Definition Template ... 351
Creating a service ... 352
Step 1: Configuring Basic Service Parameters ... 353
Step 2: Configure a Network ... 355
Step 3: Configuring Your Service to Use a Load Balancer ... 356
Step 4: Configuring Your Service to Use Service Discovery ... 360
Step 5: Configuring Your Service to Use Service Auto Scaling ... 361
Step 6: Review and Create Your Service ... 363
Updating a Service ... 363
Deleting a Service ... 366
Deployment Types ... 366
Rolling Update ... 367
Blue/Green Deployment with CodeDeploy ... 367
External Deployment ... 371
Service Load Balancing ... 376
Service Load Balancing Considerations ... 377
Load Balancer Types ... 379
Creating a Load Balancer ... 382
Registering Multiple Target Groups with a Service ... 393
Service Auto Scaling ... 395
IAM Permissions Required for Service Auto Scaling ... 395
Target Tracking Scaling Policies ... 396
Step Scaling Policies ... 401
Service Discovery ... 403
Service Discovery Concepts ... 404

Service Discovery Considerations ... 404
Amazon ECS Console Experience ... 405
Service Discovery Pricing ... 405
Service Throttle Logic ... 406
Resources and Tags ... 407
Tagging Your Resources ... 407
Tag Basics ... 407
Tagging Your Resources ... 408
Tag Restrictions ... 408
Tagging Your Resources for Billing ... 409
Working with Tags Using the Console ... 409
Working with Tags Using the CLI or API ... 411
Usage Reports ... 413
Monitoring ... 414
Monitoring Tools ... 414
Automated Tools ... 415
Manual Tools ... 415
CloudWatch Metrics ... 416
Enabling CloudWatch Metrics ... 416
Available Metrics and Dimensions ... 416
Cluster Reservation ... 419
Cluster Utilization ... 420
Service Utilization ... 421
Service `RUNNING` Task Count ... 422
Viewing Amazon ECS Metrics ... 422
Tutorial: Scaling with CloudWatch Alarms ... 424
Events and EventBridge ... 428
Amazon ECS Events ... 428
Handling Events ... 437
CloudWatch Container Insights ... 439
Container Insights Considerations ... 439
Working with Container Insights-enabled clusters ... 440
Logging Amazon ECS API Calls with AWS CloudTrail ... 441
Amazon ECS Information in CloudTrail ... 441
Understanding Amazon ECS Log File Entries ... 442
Security ... 444
Identity and Access Management ... 444
Audience ... 445
Authenticating With Identities ... 445
Managing Access Using Policies ... 447
How Amazon Elastic Container Service Works with IAM ... 448
Identity-Based Policy Examples ... 452
Supported Resource-Level Permissions ... 464
Managed Policies and Trust Relationships ... 465
Service-Linked Role ... 473
Task Execution IAM Role ... 482
Container Instance IAM Role ... 486
IAM Roles for Tasks ... 489
CodeDeploy IAM Role ... 493
CloudWatch Events IAM Role ... 496
Troubleshooting ... 499
Logging and Monitoring ... 501
Compliance Validation ... 502
Infrastructure Security ... 503
Interface VPC endpoints (AWS PrivateLink) ... 503
Using the Amazon ECS CLI ... 506
Installing the Amazon ECS CLI ... 506

Step 1: Download the Amazon ECS CLI ... 506
Step 2: Verify the Amazon ECS CLI ... 506
Step 3: Apply Execute Permissions to the Binary ... 511
Step 4: Complete the Installation ... 511
Configuring the Amazon ECS CLI ... 512
Profiles ... 512
Cluster Configurations ... 512
Order of Precedence ... 512
Migrating Configuration Files ... 513
Migrating Older Configuration Files to the v1.0.0+ Format ... 514
Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI ... 514
Prerequisites ... 514
Step 1: Create the Task Execution IAM Role ... 515
Step 2: Configure the Amazon ECS CLI ... 515
Step 3: Create a Cluster and Configure the Security Group ... 516
Step 4: Create a Compose File ... 516
Step 5: Deploy the Compose File to a Cluster ... 517
Step 6: View the Running Containers on a Cluster ... 517
Step 7: View the Container Logs ... 517
Step 8: Scale the Tasks on the Cluster ... 518
Step 9: View your Web Application ... 518
Step 10: Clean Up ... 518
Tutorial: Creating a Cluster with an EC2 Task Using the Amazon ECS CLI ... 519
Prerequisites ... 519
Step 1: Configure the Amazon ECS CLI ... 519
Step 2: Create Your Cluster ... 519
Step 3: Create a Compose File ... 520
Step 4: Deploy the Compose File to a Cluster ... 520
Step 5: View the Running Containers on a Cluster ... 520
Step 6: Scale the Tasks on a Cluster ... 521
Step 7: Create an ECS Service from a Compose File ... 521
Step 8: View your Web Application ... 521
Step 9: Clean Up ... 522
Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI ... 522
Prerequisites ... 522
Configure the Amazon ECS CLI ... 522
Create an Amazon ECS Service Configured to Use Service Discovery ... 523
Amazon ECS Command Line Reference ... 525
ecs-cli ... 526
ecs-cli configure ... 527
ecs-cli up ... 533
ecs-cli down ... 541
ecs-cli scale ... 543
ecs-cli ps ... 544
ecs-cli push ... 546
ecs-cli pull ... 548
ecs-cli images ... 550
ecs-cli license ... 553
ecs-cli compose ... 554
ecs-cli compose service ... 566
ecs-cli logs ... 588
ecs-cli check-attributes ... 590
ecs-cli registry-creds ... 592
ecs-cli local ... 598
Using Docker Compose File Syntax ... 605
Using Amazon ECS Parameters ... 606
Service Quotas ... 612

Adjustable service quotas ... 612
Nonadjustable service quotas ... 612
Using Service Quotas ... 613
Common Use Cases ... 615
Microservices ... 615
Auto Scaling ... 615
Service Discovery ... 616
Authorization and Secrets Management ... 616
Logging ... 616
Continuous Integration and Continuous Deployment ... 616
Batch Jobs ... 617
Savings Plans ... 618
Amazon Elastic Container Service on AWS Outposts ... 619
Prerequisites ... 619
Limitations ... 619
Network Connectivity Considerations ... 619
Creating an Amazon ECS Cluster on an Outpost ... 619
Getting started with AWS App Mesh and Amazon ECS ... 623
Scenario ... 623
Prerequisites ... 623
Step 1: Create a mesh and virtual service ... 624
Step 2: Create a virtual node ... 624
Step 3: Create a virtual router and route ... 625
Step 4: Review and create ... 627
Step 5: Create additional resources ... 627
Step 6: Update services ... 631
AWS Deep Learning Containers on Amazon ECS ... 642
Deep Learning Containers with Elastic Inference on Amazon ECS ... 642
Tutorials ... 643
Tutorial: Creating a VPC ... 643
Step 1: Create an Elastic IP Address for Your NAT Gateway ... 643
Step 2: Run the VPC Wizard ... 643
Step 3: Create Additional Subnets ... 644
Next Steps ... 645
Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI ... 645
Prerequisites ... 645
Step 1: Create a Cluster ... 646
Step 2: Register a Task Definition ... 646
Step 3: List Task Definitions ... 647
Step 4: Create a Service ... 647
Step 5: List Services ... 648
Step 6: Describe the Running Service ... 648
Step 7: Clean Up ... 650
Tutorial: Creating a Cluster with an EC2 Task Using the AWS CLI ... 650
Prerequisites ... 650
Step 1: Create a Cluster ... 651
Step 2: Launch an Instance with the Amazon ECS AMI ... 651
Step 3: List Container Instances ... 651
Step 4: Describe your Container Instance ... 652
Step 5: Register a Task Definition ... 654
Step 6: List Task Definitions ... 655
Step 7: Run a Task ... 655
Step 8: List Tasks ... 656
Step 9: Describe the Running Task ... 656
Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets ... 657
Prerequisites ... 657
Step 1: Create an Secrets Manager Secret ... 657

Step 2: Update Your Task Execution IAM Role ... 658
Step 3: Create an Amazon ECS Task Definition ... 658
Step 4: Create an Amazon ECS Cluster ... 659
Step 5: Run an Amazon ECS Task ... 660
Step 6: Verify ... 660
Step 7: Clean Up ... 661
Tutorial: Creating a Service Using Service Discovery ... 662
Prerequisites ... 663
Step 1: Create the Service Discovery Resources ... 663
Step 2: Create the Amazon ECS Resources ... 664
Step 3: Verify Service Discovery ... 667
Step 4: Clean Up ... 670
Tutorial: Creating a Service Using a Blue/Green Deployment ... 671
Prerequisites ... 671
Step 1: Create an Application Load Balancer ... 672
Step 2: Create an Amazon ECS Cluster ... 673
Step 3: Register a Task Definition ... 673
Step 4: Create an Amazon ECS Service ... 674
Step 5: Create the AWS CodeDeploy Resources ... 675
Step 6: Create and Monitor an CodeDeploy Deployment ... 676
Step 7: Clean Up ... 679
Tutorial: Listening for Amazon ECS CloudWatch Events ... 680
Prerequisite: Set Up a Test Cluster ... 680
Step 1: Create the Lambda Function ... 680
Step 2: Register Event Rule ... 681
Step 3: Test Your Rule ... 681
Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events ... 682
Prerequisite: Set Up a Test Cluster ... 682
Step 1: Create and Subscribe to an Amazon SNS Topic ... 682
Step 2: Register Event Rule ... 682
Step 3: Test Your Rule ... 683
Tutorial: Using Amazon EFS ... 684
Step 1: Create an Amazon ECS cluster ... 684
Step 2: Create a security group for the Amazon EFS file system ... 685
Step 3: Create an Amazon EFS file system ... 686
Step 4: Add content to the Amazon EFS file system ... 686
Step 5: Create a task definition ... 688
Step 6: Run a task and view the results ... 688
Troubleshooting ... 690
Checking stopped tasks for errors ... 690
CannotPullContainer task errors ... 692
Service Event Messages ... 694
Service Event Messages ... 694
Invalid CPU or memory value specified ... 697
`CannotCreateContainerError: API error (500): devmapper` ... 698
Troubleshooting service load balancers ... 699
Enabling Docker debug output ... 700
Amazon ECS Log File Locations ... 701
Amazon ECS Container Agent Log ... 701
Amazon ECS `ecs-init` Log ... 703
IAM Roles for Tasks Credential Audit Log ... 703
Amazon ECS logs collector ... 704
Agent introspection diagnostics ... 705
Docker diagnostics ... 707
List Docker containers ... 707
View Docker Logs ... 708
Inspect Docker Containers ... 708

API error messages 709
Troubleshooting IAM Roles for Tasks 710
Windows containers 713
Windows container caveats 713
Getting started with Windows containers 714
Step 1: Create a Windows cluster 714
Step 2: Register a Windows task definition 716
Step 3: Create a service with your task definition 717
Step 4: View your service 717
Windows task definitions 718
Windows task definition parameters 718
Windows sample task definitions 720
Windows IAM roles for tasks 721
IAM roles for task container bootstrap script 721
Pushing Windows images to Amazon ECR 722
Using gMSAs for Windows Containers 723
Considerations 723
Prerequisites 723
Setting Up gMSA-capable Windows Containers on Amazon ECS 724
Document History 727
AWS glossary 746

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 over your infrastructure, 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. 119).

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.

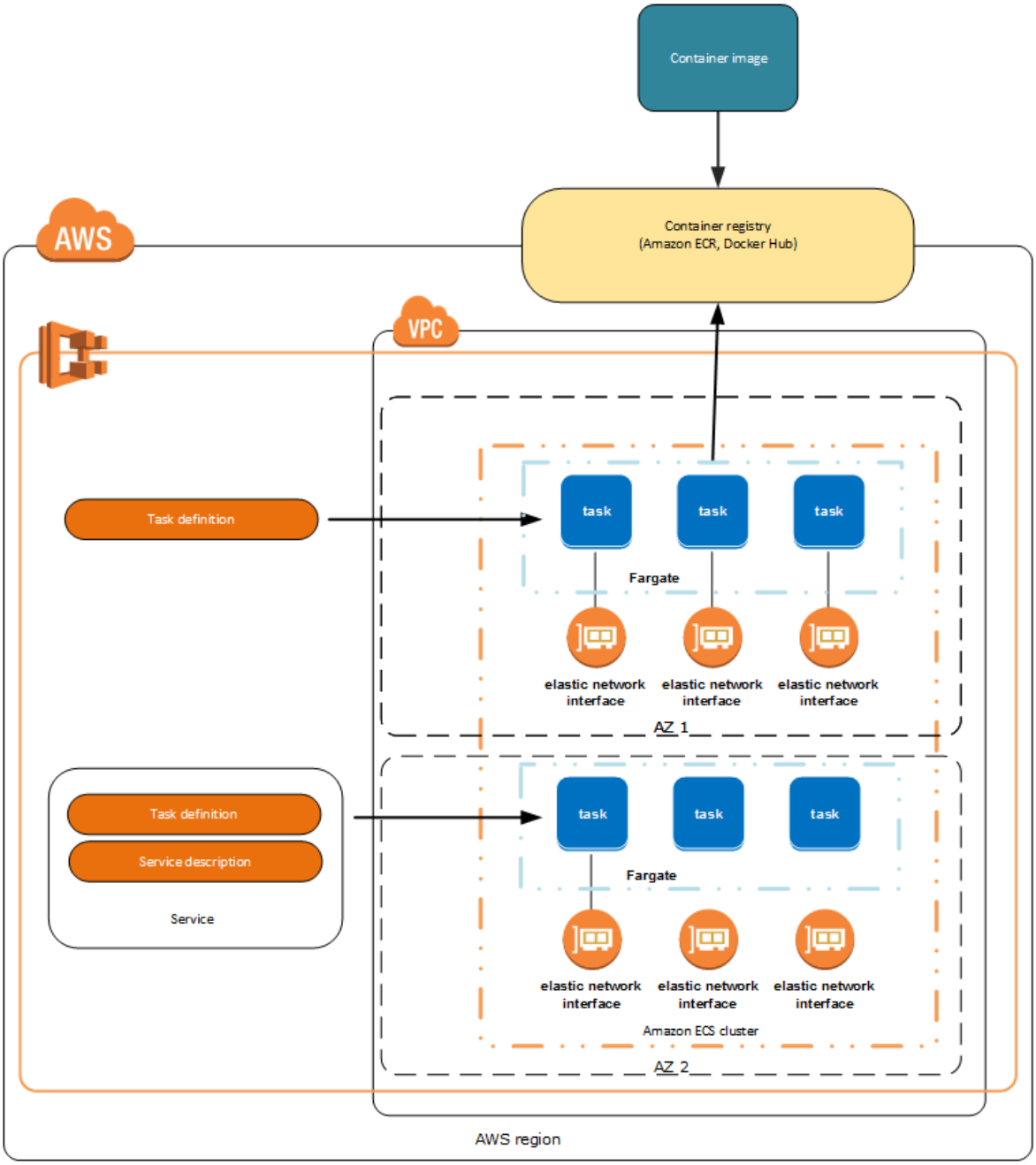
AWS Elastic Beanstalk can also be used to rapidly develop, test, and deploy Docker containers in conjunction with other components of your application infrastructure; however, using Amazon ECS directly provides more fine-grained control and access to a wider set of use cases. For more information, see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

The AWS container services team maintains a public roadmap on GitHub. It contains information about what the teams are working on and allows all AWS customers the ability to give direct feedback. For more information, see AWS Containers Roadmap.

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:

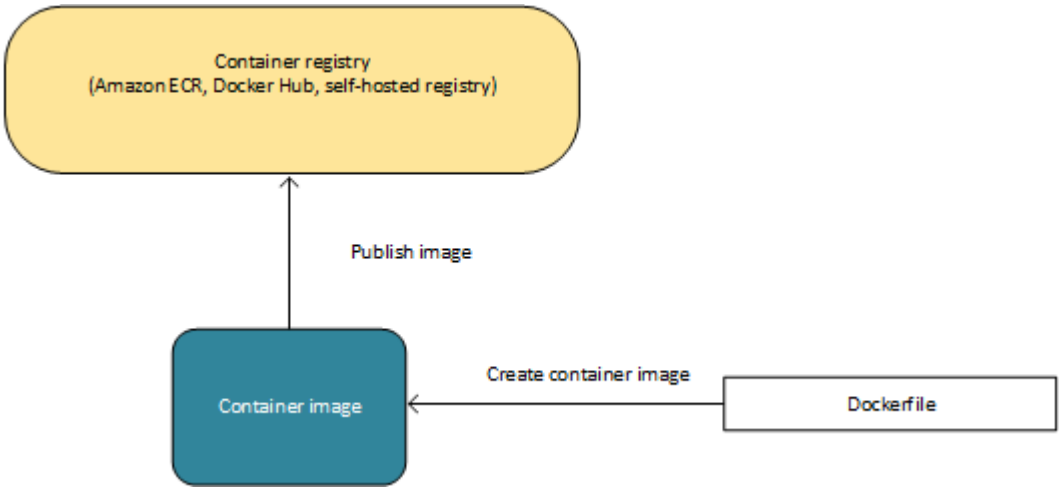


The following sections dive into these individual elements of the Amazon ECS architecture in more detail.

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. 73).

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. 177).

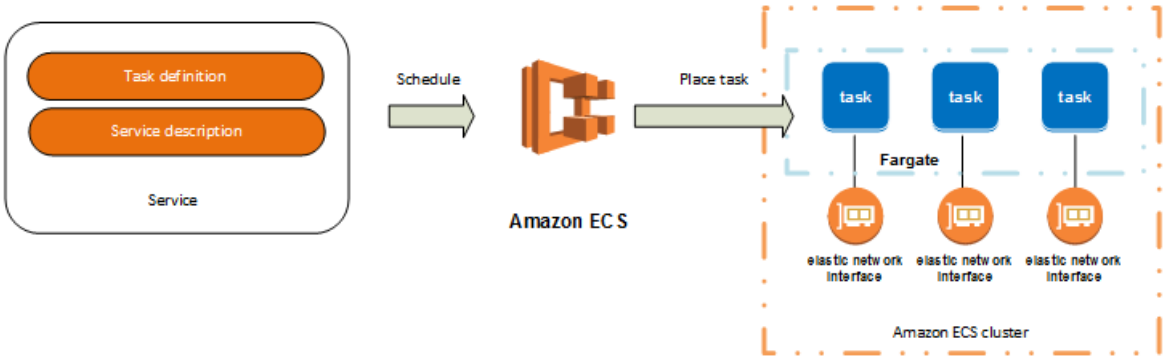
```
{
    "family": "webserver",
    "containerDefinitions": [
        {
            "name": "web",
            "image": "nginx",
            "memory": "100",
            "cpu": "99"
        },
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "memory": "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. 315).



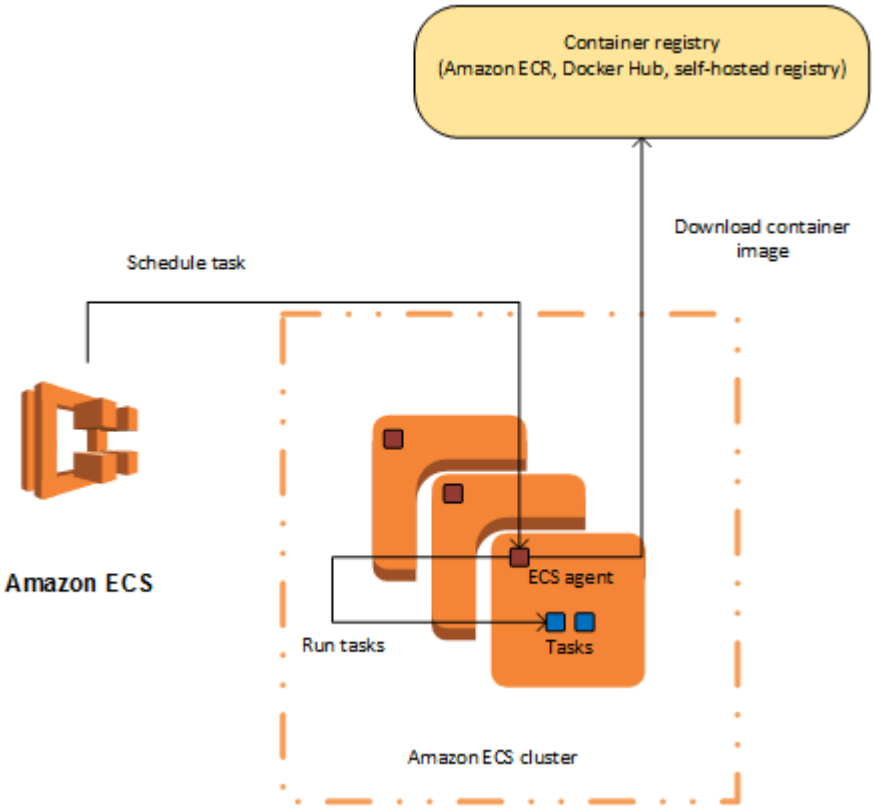
Clusters

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. 37). If you are using the EC2 launch type, you can read about creating container instances at Amazon ECS Container Instances (p. 188).

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. 250).



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 using Fargate (p. 23) 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 Identity and Access Management for Amazon Elastic Container Service (p. 444).

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. 395).

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. 376).

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 Using Amazon EC2 (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. 506).

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.

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

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on 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 and 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 administrator user for yourself and add the user to an administrators group (console)

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, choose **Users** and then choose **Add user**.
3. For **User name**, enter **`Administrator`**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **`Administrators`**.
10. Choose **Filter policies**, and then select **AWS managed -job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

    Note
    You must activate IAM user and role access to Billing before you can use the `AdministratorAccess` permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.
12. 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.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
15. 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 that restrict user permissions to specific AWS resources, see 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/
```

To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link** on the dashboard.

For more information about IAM, see the AWS Identity and Access Management User Guide.

Create a Key Pair

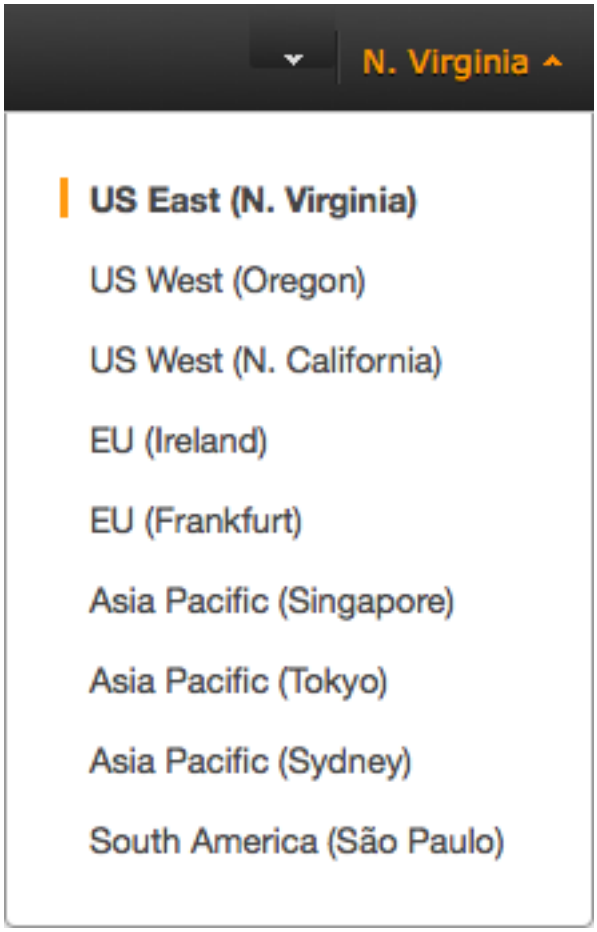
For Amazon ECS, a key pair is only needed if you intend on using the EC2 launch type.

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
```

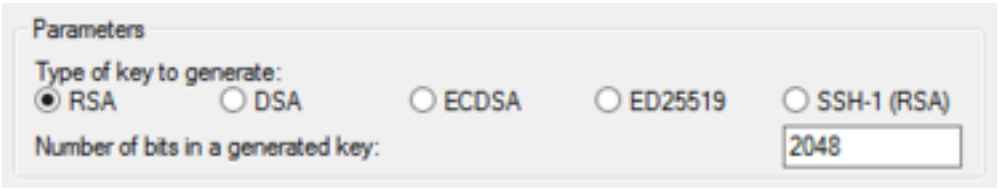
For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Linux Instances*.

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.

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.

> **Note**
> The Amazon ECS console first-run experience creates a VPC for your cluster, so if you intend to use the Amazon ECS console, you can skip to the next section.

If you have a default VPC, you also can skip this section and move to the next task, Create a Security Group (p. 12). To determine whether you have a default VPC, see Supported Platforms in the Amazon EC2 Console in the *Amazon EC2 User Guide for Linux Instances*. Otherwise, you can create a nondefault VPC in your account using the steps below.

> **Important**
> If your account supports Amazon EC2 Classic in a region, then you do not have a default VPC in that region.

To create a nondefault VPC

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. From the navigation bar, select a region for the VPC. VPCs are specific to a region, so you should select the same region in which you created your key pair.
3. On the VPC dashboard, choose **Launch VPC Wizard**.
4. On the **Step 1: Select a VPC Configuration** page, ensure that **VPC with a Single Public Subnet** is selected, and choose **Select**.
5. On the **Step 2: VPC with a Single Public Subnet** page, enter a friendly name for your VPC in the **VPC name** field. Leave the other default configuration settings, and choose **Create VPC**. On the confirmation page, choose **OK**.

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

Create a Security Group

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

> **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.
3. In the navigation pane, choose **Security Groups**, **Create Security Group**.
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.

Installing Docker on Amazon Linux 2

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 Amazon Linux 2 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`.

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

7. Log out and log back in again to pick up the new `docker` group permissions. You can accomplish this by closing your current SSH terminal window and reconnecting to your instance in a new one. Your new SSH session will have the appropriate `docker` group permissions.
8. Verify that the `ec2-user` can run Docker commands without `sudo`.

```
docker info
```

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

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

Create a Docker Image

Amazon ECS task definitions use 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 Amazon 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 Amazon ECS task definition.

To create a Docker image of a simple web application

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

   ```
   touch Dockerfile
   ```

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

   ```
   FROM ubuntu:18.04

   # Install dependencies
   RUN apt-get update && \
    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 && \
    echo 'mkdir -p /var/run/apache2' >> /root/run_apache.sh && \
    echo 'mkdir -p /var/lock/apache2' >> /root/run_apache.sh && \
    echo '/usr/sbin/apache2 -D FOREGROUND' >> /root/run_apache.sh && \
    chmod 755 /root/run_apache.sh

   EXPOSE 80

   CMD /root/run_apache.sh
   ```

   This Dockerfile uses the Ubuntu 18.04 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 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**.

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.

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

Output:

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

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

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

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

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

Output:

```
Login Succeeded
```

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

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

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

Clean up

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

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

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 the following 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 in the *Amazon Elastic Container Service Developer Guide*.
- The Amazon ECS container instance role (`ecsInstanceRole`) that you use with your container instances must possess the following IAM policy permissions for Amazon ECR.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        }
    ]
}
```

  If you use the `AmazonEC2ContainerServiceforEC2Role` managed policy for your container instances, then your role has the proper permissions. To check that your role supports Amazon ECR, see Amazon ECS Container Instance IAM Role in the *Amazon Elastic Container Service Developer Guide*.
- 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`*.

Getting Started with Amazon ECS Using Amazon EC2

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage your containers. You can host your containers on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks using the Fargate launch type. For more control, you can host your tasks on a cluster of Amazon EC2 instances that you manage by using the EC2 launch type. For a broad overview on Amazon ECS, see What is Amazon Elastic Container Service? (p. 1).

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

> **Important**
> For information about getting started with Amazon ECS using the Fargate launch type, see Getting Started with Amazon ECS using Fargate (p. 23).

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

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 the Amazon ECS First Run Wizard Permissions (p. 454) IAM policy example.

Step 1: Register 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. For more information about task definitions, see Amazon ECS Task Definitions (p. 73).

The following steps walk you through creating a task definition that will deploy a simple web application.

To register a task definition

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the Region you want to use.
3. In the navigation pane, choose **Task Definitions**, **Create new Task Definition**.
4. On the **Select launch type compatibility** page, select **EC2** and choose **Next step**.
5. On the **Select configure task and container definitions** page, choose **Configure via JSON**.
6. Copy and paste the following example task definition into the box and then choose **Save**.

```
{
    "containerDefinitions": [
```

```
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
            ],
            "cpu": 10,
            "memory": 300,
            "image": "httpd:2.4",
            "name": "simple-app"
        }
    ],
    "family": "console-sample-app-static"
}
```

7. Choose **Create**.

Step 2: Create a cluster

An Amazon ECS cluster is a logical grouping of tasks, services, and container instances. When creating a cluster using the console, Amazon ECS creates a AWS CloudFormation stack that takes care of the Amazon EC2 instance creation, networking and IAM configuration for you. For more information about clusters, see Amazon ECS Clusters (p. 37).

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

To create a cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. On the **Select cluster template** page, choose **EC2 Linux + Networking**.
6. For **Cluster name**, choose a name for your cluster.
7. In the **Instance configuration** section, do the following:

    a. For **EC2 instance type**, choose either the **t2.micro** or **t3.micro** instance type to use for the container instance. Instance types with more CPU and memory resources can handle more tasks, but that is unnecessary for this getting started guide. For more information about the different instance types, see Amazon EC2 Instances.

    b. For **Number of instances**, type **1**. Amazon EC2 instances incur costs while they exist in your AWS resources. For more information, see Amazon EC2 Pricing.

c. For **EC2 Ami Id**, use the default value which is the Amazon Linux 2 Amazon ECS-optimized AMI. For more information about the Amazon ECS-optimized AMI, see Amazon ECS-optimized AMIs (p. 190).

8. In the **Networking** section, for **VPC** choose either **Create a new VPC** to have Amazon ECS create a new VPC for the cluster to use, or choose an existing VPC to use. For more information on creating your own VPC, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters (p. 643).
9. In the **Container instance IAM role** section, choose **Create new role** to have Amazon ECS create a new IAM role for your container instances, or choose an existing Amazon ECS container instance (`ecsInstanceRole`) role that you have already created. For more information, see Amazon ECS Container Instance IAM Role (p. 486).
10. Choose **Create**.

Step 3: Create a Service

An Amazon ECS service enables you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it in order to maintain the desired number of tasks in the service. For more information on services, see Amazon ECS services (p. 338).

To create a service

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. Select the cluster you created in the previous step.
5. On the **Services** tab, choose **Create**.
6. In the **Configure service** section, do the following:
   a. For **Task definition**, select the **console-sample-app-static** task definition you created in step 1.
   b. For **Cluster**, select the cluster you created in step 2.
   c. For **Service name**, select a name for your service.
   d. For **Number of tasks**, enter `1`.
7. Use the default values for the rest of the fields and choose **Next step**.
8. In the **Configure network** section, leave the default values and choose **Next step**.
9. In the **Set Auto Scaling** section, leave the default value and choose **Next step**.
10. Review the options and choose **Create service**.
11. Choose **View service** to review your service.

Step 4: View your Service

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

To view the service details

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.

4. Select the cluster you created step 2.
5. On the **Services** tab, choose the service you created in step 3.
6. On the **Service: *service-name*** page, choose the **Tasks** tab.
7. Confirm that the task is in a **RUNNING** state. If it is, select the task to view the task details. If it is not in a **RUNNING** status, refresh the service details screen until it is.
8. In the **Containers** section, expand the container details. In the **Network bindings** section, for **External Link** you will see the **IPv4 Public IP** address to use to access the web application.
9. Enter the **IPv4 Public IP** address in your web browser and you should see a webpage that displays the **Amazon ECS sample** application.

Step 5: Clean Up

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

The Amazon ECS resources created in this getting started guide, such as the cluster and service can be cleaned up using the Amazon ECS console.

To clean up the resources

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose **Clusters**.
3. Select the cluster you created step 2.
4. On the **Services** tab, select the service you created in step 3 and choose **Delete**. At the confirmation prompt, enter **delete me** and then choose **Delete**.
5. On the cluster details page, choose **Delete cluster**. At the confirmation prompt, enter **delete me** and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including the VPC and Amazon EC2 instances.

Getting Started with Amazon ECS using Fargate

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage your containers. You can host your containers on a serverless infrastructure that is managed by Amazon ECS by launching your services or tasks on AWS Fargate. For a broad overview on Amazon ECS, see What is Amazon Elastic Container Service? (p. 1).

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks. In the Regions where Amazon ECS supports AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using the Fargate 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 getting started with Amazon ECS using the EC2 launch type, see Getting Started with Amazon ECS Using Amazon EC2 (p. 19).

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

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 Permissions (p. 454) IAM policy example.

The first-run wizard attempts to automatically create the task execution IAM role, which is required for Fargate tasks. To ensure that the first-run experience is able to create this IAM role, 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 the task execution role so that it is available on the account to be used. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

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.

1. Open the Amazon ECS console first-run wizard at https://console.aws.amazon.com/ecs/home#/firstRun.

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 Amazon ECS on AWS Fargate (p. 27).

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. 85).

   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. 84).

   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. 376).

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: 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.

Step 6: Clean Up

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

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

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, select the cluster to delete.
4. Choose **Delete Cluster**. At the confirmation prompt, enter **delete me** and then choose **Delete**. Deleting the cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

Amazon ECS on AWS Fargate

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.

Amazon ECS on AWS Fargate is supported in the following Regions. The supported Availability Zone IDs are noted when applicable.

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1 (`aps1-az1` & `aps1-az3` only)
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1

Region Name	Region
Middle East (Bahrain)	me-south-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

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

- *Getting Started with Amazon ECS using Fargate* (p. 23)
- the section called "Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI" (p. 645)
- the section called "Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI" (p. 514)

Task Definitions

Amazon ECS tasks on Fargate 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. 103).
- `volumes` – Fargate tasks only support bind mount host volumes, so the `dockerVolumeConfiguration` parameter is not supported. For more information, see Volumes (p. 110).

To ensure that your task definition validates for use with Fargate, you can specify the following when you register the task definition:

- In the AWS Management Console, for the **Requires Compatibilities** field, specify `FARGATE`.
- In the AWS CLI, specify the `--requires-compatibilities` option.

- In the Amazon ECS API, specify the `requiresCompatibilities` flag.

Network Mode

Amazon ECS task definitions for Fargate 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. 141).

A network configuration is also required when creating a service or manually running tasks. For more information, see Task Networking (p. 32).

Task CPU and Memory

Amazon ECS task definitions for Fargate 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.

CPU value	Memory value
256 (.25 vCPU)	0.5 GB, 1 GB, 2 GB
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2048 (2 vCPU)	Between 4 GB and 16 GB in 1-GB increments
4096 (4 vCPU)	Between 8 GB and 30 GB in 1-GB increments

Task Resource Limits

Amazon ECS task definitions for Fargate support the `ulimits` parameter to define the resource limits to set for a container.

Fargate tasks use the default resource limit values with the exception of the `nofile` resource limit parameter, which Fargate overrides. The `nofile` resource limit sets a restriction on the number of open files that a container can use. The default `nofile` soft limit is `1024` and hard limit is `4096` for Fargate tasks. These limits can be adjusted in a task definition if your tasks needs to handle a larger number of files. The following shows a snippet of a task definition where the `nofile` limit has been doubled:

```
"ulimits": [
    {
       "name": "nofile",
       "softLimit": 2048,
       "hardLimit": 8192
    }
]
```

For more information on the other resource limits that can be adjusted, see Resource Limits (p. 102).

Logging

Amazon ECS task definitions for Fargate support the `awslogs`, `splunk`, `firelens`, and `fluentd` log drivers for the log configuration.

The `awslogs` log driver configures your Fargate tasks to send log information to Amazon CloudWatch Logs. The following shows a snippet of a task definition where the awslogs log driver is configured:

```
"logConfiguration": {
   "logDriver": "awslogs",
   "options": {
      "awslogs-group" : "/ecs/fargate-task-definition",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
}
```

For more information about using the `awslogs` log driver in a task definition to send your container logs to CloudWatch Logs, see Using the awslogs Log Driver (p. 143).

For more information about the `firelens` log driver in a task definition, see Custom Log Routing (p. 150).

For more information about using the `splunk` log driver in a task definition, see Example: `splunk` Log Driver (p. 178).

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. 482).

Example Task Definition

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

```
{
   "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"
            }
        ]
    }
  ],
  "cpu": "256",
  "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
  "family": "fargate-task-definition",
  "memory": "512",
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
      "FARGATE"
   ]
}
```

Task Storage

For Fargate tasks, the following storage types are supported:

- Amazon EFS volumes for persistent storage. For more information, see Amazon EFS Volumes (p. 138).
- Ephemeral storage for nonpersistent storage.

When provisioned, each Amazon ECS task on Fargate receives the following ephemeral storage. The ephemeral storage configuration depends on which platform version the task is using. After a Fargate task stops, the ephemeral storage is deleted. For more information about Amazon ECS default service limits, see Amazon ECS service quotas (p. 612).

Fargate tasks using platform version 1.4.0 or later

All Amazon ECS on Fargate tasks using platform version 1.4.0 or later receive a minimum of 20 GB of ephemeral storage. The amount of storage is not adjustable.

For tasks using platform version 1.4.0 or later that are launched on May 28, 2020 or later, the ephemeral storage is encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key.

Fargate tasks using platform version 1.3.0 or earlier

For Amazon ECS on Fargate tasks using platform version 1.3.0 or earlier, each task receives the following ephemeral storage.

- 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.

Tasks and Services

After you have your Amazon ECS task definitions for Fargate prepared, there are some decisions to make when creating your service.

Task Networking

Amazon ECS tasks for Fargate require the `awsvpc` network mode, which provides each task with an elastic network interface. When you run a task or create a service with this network mode, you must specify one or more subnets to attach the network interface and one or more security groups to apply to the network interface.

If you are using public subnets, decide whether to provide a public IP address for the network interface. For a Fargate task in a public subnet to pull container images, a public IP address needs to be assigned to the task's elastic network interface, with a route to the internet or a NAT gateway that can route requests to the internet. For a Fargate task in a private subnet to pull container images, 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. 141).

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. 376).

Private Registry Authentication

Amazon ECS tasks for Fargate 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:

```
"containerDefinitions": [
   {
      "image": "private-repo/private-image",
      "repositoryCredentials": {
         "credentialsParameter:
 "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name"
      }
   }
]
```

For more information, see Private Registry Authentication for Tasks (p. 160).

Clusters

Clusters may 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. 37).

Fargate Spot

Amazon ECS capacity providers enable you to use both Fargate and Fargate Spot capacity with your Amazon ECS tasks.

With Fargate Spot you can run interruption tolerant Amazon ECS tasks at a discounted rate compared to the Fargate price. Fargate Spot runs tasks on spare compute capacity. When AWS needs the capacity back, your tasks will be interrupted with a two-minute warning. For more information, see Using AWS Fargate Capacity Providers (p. 67).

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. 334).

Fargate Savings Plans

Savings Plans are a pricing model that offer significant savings on AWS usage. You commit to a consistent amount of usage, in USD per hour, for a term of 1 or 3 years, and receive a lower price for that usage. For more information, see the Savings Plans User Guide.

To create a Savings Plan for your Fargate usage, use the **Compute Savings Plans** type. For more information, see Savings Plans and Amazon ECS (p. 618).

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. 334).

Topics

- Platform version considerations (p. 34)
- Available AWS Fargate platform versions (p. 34)
- Migrating from platform version 1.3.0 to 1.4.0 (p. 36)

Platform version considerations

The following should be considered when specifying a platform version:

- When specifying a platform version, you can use either a specific version number, for example `1.4.0`, or `LATEST` (which uses the `1.3.0` platform version).
- To use a specific platform version, specify the version number when creating or updating your service. If you specify `LATEST`, your tasks use platform version `1.3.0`.
- In the China (Beijing) and China (Ningxia) Regions, the only supported platform versions are `1.4.0` and `1.3.0`. The AWS Management Console displays older platform versions but an error will be returned if they are chosen. The `LATEST` platform version is supported because it uses the `1.3.0` 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. 363).
- 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-1.4.0

- Beginning on May 28, 2020, any new Fargate task that is launched using platform version 1.4.0 will have its ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key. For more information, see Fargate Task Storage (p. 125).
- Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see Amazon EFS Volumes (p. 138).
- The ephemeral task storage has been increased to a minimum of 20 GB for each task. For more information, see Fargate Task Storage (p. 125).
- The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4.0, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see Fargate Task Networking in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- Task ENIs add support for jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames will reduce overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC.
- CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).

- Added support for the task metadata endpoint version 4 which provides additional information for your Fargate tasks, including network stats for the task and which Availability Zone the task is running in. For more information, see Task Metadata Endpoint version 4 (p. 292).
- Added support for the `SYS_PTRACE` Linux parameter in container definitions. For more information, see Linux Parameters (p. 103).
- The Fargate container agent replaces the use of the Amazon ECS container agent for all Fargate tasks. This change should not have an effect on how your tasks run.
- The container runtime is now using Containerd instead of Docker. This change should not have an effect on how your tasks run. You will notice that some error messages that originate with the container runtime will change from mentioning Docker to more general errors. For more information, see Stopped tasks error codes in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Fargate platform version-1.3.0

- Beginning on Sept 30, 2019, any new Fargate task that is launched supports the `awsfirelens` log driver. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics. For more information, see Custom Log Routing (p. 150).
- 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. 336).
- 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. 116), Container Dependency (p. 107), and Container Timeouts (p. 108).
- 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. 163).
- 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. 163).
- 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. 97).
- Beginning on July 9, 2019, any new Fargate tasks that is launched supports CloudWatch Container Insights. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).
- Beginning on December 3, 2019, the Fargate Spot capacity provider is supported. For more information, see Using AWS Fargate Capacity Providers (p. 67).

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. 160).

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. 292).
- Added support for Docker health checks in container definitions. For more information, see Health Check (p. 90).
- Added support for Amazon ECS service discovery. For more information, see Service Discovery (p. 403).

Fargate Platform Version-1.0.0

- Based on Amazon Linux 2017.09.
- Initial release.

Migrating from platform version 1.3.0 to 1.4.0

The following should be considered when migrating your Amazon ECS on Fargate tasks from platform version 1.3.0 to platform version 1.4.0.

- The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4.0, all Amazon ECS on Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see Fargate Task Networking in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- If you are using interface VPC endpoints, the following should be considered.
  - When using container images hosted with Amazon ECR, both the **com.amazonaws.region.ecr.dkr** and **com.amazonaws.region.ecr.api** Amazon ECR VPC endpoints as well as the Amazon S3 gateway endpoint are required. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.
  - When using a task definition that references Secrets Manager secrets to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Secrets Manager. For more information, see Using Secrets Manager with VPC Endpoints in the *AWS Secrets Manager User Guide*.
  - When using a task definition that references Systems Manager Parameter Store parameters to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Systems Manager. For more information, see Using Systems Manager with VPC endpoints in the *AWS Systems Manager User Guide*.
  - Ensure that the security group in the Elastic Network Interface (ENI) associated with your task has the security group rules created to allow traffic between the task and the VPC endpoints you are using.

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. If you are using capacity providers, a cluster is also a logical grouping of capacity providers. 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.

Topics

- Cluster Concepts (p. 37)
- Creating a Cluster (p. 38)
- Amazon ECS Cluster Capacity Providers (p. 40)
- Amazon ECS Cluster Auto Scaling (p. 42)
- Using AWS Fargate Capacity Providers (p. 67)
- Updating Cluster Settings (p. 71)
- Deleting a Cluster (p. 71)

Cluster Concepts

The following are general concepts about Amazon ECS clusters.

- Clusters are Region-specific.
- The following are the possible states that a cluster can be in.

  ACTIVE

  The cluster is ready to accept tasks and, if applicable, you can register container instances with the cluster.

  PROVISIONING

  The cluster has capacity providers associated with it and the resources needed for the capacity provider are being created.

  DEPROVISIONING

  The cluster has capacity providers associated with it and the resources needed for the capacity provider are being deleted.

  FAILED

  The cluster has capacity providers associated with it and the resources needed for the capacity provider have failed to create.

  INACTIVE

  The cluster has been deleted. Clusters with an `INACTIVE` status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so you should not rely on `INACTIVE` clusters persisting.
- A cluster may contain a mix of tasks using either the Fargate or EC2 launch types. For more information about launch types, see Amazon ECS Launch Types (p. 119).
- A cluster may contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers, however when specifying a capacity provider strategy they may only contain one or the other but not both. For more information, see Amazon ECS Cluster Capacity Providers (p. 40).

- For tasks using the EC2 launch type, clusters can contain multiple different container instance types, but each container instance may only be registered to one cluster at a time.
- Custom IAM policies may be created to allow or restrict user access to specific clusters. For more information, see the Cluster Examples (p. 458) section in Amazon Elastic Container Service Identity-Based Policy Examples (p. 452).

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). You can register Amazon EC2 instances during cluster creation or register additional instances with the cluster after creating it.

The console cluster creation wizard provides a simple way to create the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack. It also lets you customize several common cluster configuration options. However, the 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.

If you add or modify the underlying cluster resources directly after they are created by the wizard you may receive an error when attempting to delete the cluster. AWS CloudFormation refers to this as *stack drift*. For more information on detecting drift on an existing AWS CloudFormation stack, see Detect Drift on an Entire CloudFormation Stack in the *AWS CloudFormation User Guide*.

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 with a new VPC to use for Fargate tasks. The `FARGATE` and `FARGATE_SPOT` capacity providers will be automatically associated with the cluster. For more information, see Using AWS Fargate Capacity Providers (p. 67).

     You can run 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 run a task with a Fargate-compatible task definition, Fargate launches the containers for you.
   - **EC2 Linux + Networking**– With this option you can launch a cluster of tasks using the EC2 launch type using 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. 713).

Using The Networking only Template

If you chose the **Networking only** cluster template, complete the following steps. Otherwise, skip to Using the EC2 Linux + Networking or EC2 Windows + Networking template (p. 39).

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. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).
5. Choose **Create**.

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

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 GPU workloads, choose an instance type from the P2 or P3 instance family. For more information, see Working with GPUs on Amazon ECS (p. 121).
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. 190).

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. 190).
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
       The IAM role you use must have the `AmazonEC2ContainerServiceforEC2Role` managed policy attached to it, otherwise you will receive an error during cluster creation. 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. 486).

    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. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).
    i. Choose **Create**.

Amazon ECS Cluster Capacity Providers

Amazon ECS cluster capacity providers determine the infrastructure to use for your tasks. Each cluster has one or more capacity providers and an optional default capacity provider strategy. The capacity provider strategy determines how the tasks are spread across the capacity providers. When you run a

task or create a service, you may either use the cluster's default capacity provider strategy or specify a capacity provider strategy that overrides the cluster's default strategy.

Cluster Capacity Provider Concepts

Cluster capacity providers consist of the following components.

Capacity provider

A *capacity provider* is used in association with a cluster to determine the infrastructure that a task runs on.

For Amazon ECS on AWS Fargate users, the `FARGATE` and `FARGATE_SPOT` capacity providers are provided automatically. For more information, see Using AWS Fargate Capacity Providers (p. 67).

For Amazon ECS on Amazon EC2 users, a capacity provider consists of a name, an Auto Scaling group, and the settings for managed scaling and managed termination protection. This type of capacity provider is used in cluster auto scaling. For more information, see Auto Scaling group Capacity Providers (p. 42).

One or more capacity providers are specified in a capacity provider strategy, which is then associated with a cluster.

Capacity provider strategy

A *capacity provider strategy* gives you control over how your tasks use one or more capacity providers.

When you run a task or create a service, you specify a capacity provider strategy. A capacity provider strategy consists of one or more capacity providers with an optional *base* and *weight* specified for each provider.

The *base* value designates how many tasks, at a minimum, to run on the specified capacity provider. Only one capacity provider in a capacity provider strategy can have a base defined.

The *weight* value designates the relative percentage of the total number of launched tasks that should use the specified capacity provider. For example, if you have a strategy that contains two capacity providers, and both have a weight of `1`, then when the base is satisfied, the tasks will be split evenly across the two capacity providers. Using that same logic, if you specify a weight of `1` for *capacityProviderA* and a weight of `4` for *capacityProviderB*, then for every one task that is run using *capacityProviderA*, four tasks would use *capacityProviderB*.

Default capacity provider strategy

A *default capacity provider strategy* is associated with each Amazon ECS cluster. This determines the capacity provider strategy the cluster will use if no other capacity provider strategy or launch type is specified when running a task or creating a service.

Cluster Capacity Provider Considerations

The following should be considered when using cluster capacity providers:

- When you specify a capacity provider strategy, the number of capacity providers that can be specified is limited to six.
- A cluster may contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers, however when specifying a capacity provider strategy they may only contain one or the other but not both.

- A cluster may contain a mix of tasks and services using both capacity providers and launch types. A service may also be updated to use a capacity provider strategy rather than a launch type, however you must force a new deployment when doing so.
- When you specify a capacity provider strategy, the `base` value is only supported when running tasks. When creating a service, the capacity provider strategy `base` parameter is not supported.
- When using managed termination protection, managed scaling must also be used otherwise managed termination protection will not work.
- Using cluster capacity providers is not supported when using the blue/green deployment type for your services.
- Using cluster capacity providers is not supported when using Classic Load Balancers for your services.

Amazon ECS Cluster Auto Scaling

Amazon ECS cluster auto scaling enables you to have more control over how you scale tasks within a cluster. Each cluster has one or more capacity providers and an optional default capacity provider strategy. The capacity providers determine the infrastructure to use for the tasks, and the capacity provider strategy determines how the tasks are spread across the capacity providers. When you run a task or create a service, you may either use the cluster's default capacity provider strategy or specify a capacity provider strategy that overrides the cluster's default strategy. For more information about cluster capacity providers, see Amazon ECS Cluster Capacity Providers (p. 40).

Topics

- Cluster Auto Scaling Considerations (p. 42)
- Auto Scaling group Capacity Providers (p. 42)
- Tutorial: Using Cluster Auto Scaling with the AWS Management Console (p. 46)
- Tutorial: Using Cluster Auto Scaling with the AWS CLI (p. 52)

Cluster Auto Scaling Considerations

The following should be considered when using cluster auto scaling:

- The Amazon ECS service-linked IAM role is required to use cluster auto scaling. For more information, see Service-Linked Role for Amazon ECS (p. 473).
- When using capacity providers with Auto Scaling groups, the `autoscaling:CreateOrUpdateTags` permission is needed on the IAM user creating the capacity provider. This is because Amazon ECS adds a tag to the Auto Scaling group when it associates it with the capacity provider.

  Important
  Ensure any tooling you use does not remove the `AmazonECSManaged` tag from the Auto Scaling group. If this tag is removed, Amazon ECS is not able to manage it when scaling your cluster.

Auto Scaling group Capacity Providers

Amazon ECS capacity providers use Auto Scaling groups to manage the Amazon EC2 instances registered to their clusters.

Topics

- Auto Scaling group Capacity Providers Considerations (p. 43)
- Using Managed Scaling (p. 43)

- Creating an Auto Scaling group (p. 43)
- Creating a Capacity Provider (p. 44)
- Creating a Cluster (p. 45)

Auto Scaling group Capacity Providers Considerations

The following should be considered when using Auto Scaling group capacity providers.

- It is recommended that you create a new Auto Scaling group to use with a capacity provider rather than using an existing one. If you use an existing Auto Scaling group, any Amazon EC2 instances associated with the group that were already running and registered to an Amazon ECS cluster prior to the Auto Scaling group being used to create a capacity provider may not be properly registered with the capacity provider. This may cause issues when using the capacity provider in a capacity provider strategy. The DescribeContainerInstances API can confirm that a container instance is associated with a capacity provider.
- An Auto Scaling group must have a `MaxSize` greater than zero to scale out.
- When using managed termination protection, managed scaling must also be used otherwise managed termination protection will not work.

Using Managed Scaling

When creating a capacity provider, you can optionally enable managed scaling. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group. On your behalf, Amazon ECS creates an AWS Auto Scaling scaling plan with a target tracking scaling policy based on the target capacity value you specify. Amazon ECS then associates this scaling plan with your Auto Scaling group. For each of the capacity providers with managed scaling enabled, an Amazon ECS managed CloudWatch metric with the prefix `AWS/ECS/ManagedScaling` is created along with two CloudWatch alarms. The CloudWatch metrics and alarms are used to monitor the container instance capacity in your Auto Scaling groups and will trigger the Auto Scaling group to scale in and scale out as needed.

Managed scaling is only supported in Regions that AWS Auto Scaling is available in. For a list of supported Regions, see AWS Auto Scaling Regions and Endpoints in the *Amazon Web Services General Reference*.

Creating an Auto Scaling group

When creating an Auto Scaling group, you use either a launch template or launch configuration. The launch template or launch configuration specifies the Amazon EC2 instance configuration, including the AMI, the instance type, a key pair, security groups, and the other parameters that you use to launch Amazon EC2 instances.

If you use the Amazon ECS console **Create Cluster** wizard with the **EC2 Linux + Networking** option, then Amazon ECS creates an Amazon EC2 Auto Scaling launch configuration and Auto Scaling group on your behalf as part of the AWS CloudFormation stack. They are prefixed with `EC2ContainerService-<ClusterName>`, which makes them easy to identify. That Auto Scaling group can then be used in a capacity provider for that cluster.

The following should be considered when creating an Auto Scaling group for a capacity provider.

- If managed termination protection is enabled when you create a capacity provider, the Auto Scaling group and each Amazon EC2 instance in the Auto Scaling group must have instance protection from scale in enabled as well. For more information, see Instance Protection in the *AWS Auto Scaling User Guide*.

- If managed scaling is enabled when you create a capacity provider, the Auto Scaling group desired count can be set to `0`. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group.

For more information on creating an Amazon EC2 Auto Scaling launch configuration, see Launch Configurations in the *Amazon EC2 Auto Scaling User Guide*.

For more information on creating an Amazon EC2 Auto Scaling launch template, see Launch Templates in the *Amazon EC2 Auto Scaling User Guide*.

For more information on creating an Amazon EC2 Auto Scaling launch template, see Auto Scaling groups in the *Amazon EC2 Auto Scaling User Guide*.

Creating a Capacity Provider

A *capacity provider* is used in association with a cluster to determine the infrastructure that a task runs on. When creating a capacity provider, you specify the following details:

- An Auto Scaling group Amazon Resource Name (ARN)
- Whether or not to enable managed scaling. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling plans. When managed scaling is disabled, you manage your Auto Scaling groups yourself.
- Whether or not to enable managed termination protection. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled.

To create a capacity provider (AWS Management Console)

Use the following steps to create a new capacity provider for an existing Amazon ECS cluster.

To create a capacity provider

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the Region your cluster exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : *name*** page, choose **Capacity Providers**, and then choose **Create**.
6. For **Capacity provider name**, enter a capacity provider name.
7. For **Auto Scaling group**, select the Auto Scaling group to associate with the capacity provider. The Auto Scaling group must already be created. For more information, see Creating an Auto Scaling group (p. 43).
8. For **Managed scaling**, choose your managed scaling option. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling plans. When managed scaling is disabled, you manage your Auto Scaling groups yourself.
9. For **Target capacity %**, if managed scaling is enabled, specify an integer between `1` and `100`. The target capacity value is used as the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy. This target capacity value is matched on a best effort basis. For example, a value of `100` will result in the Amazon EC2 instances in your Auto Scaling group being completely utilized and any instances not running any tasks will be scaled in, but this behavior is not guaranteed at all times.

10. For **Managed termination protection**, choose your managed termination protection option. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled and if managed scaling is enabled. Managed termination protection is only supported on standalone tasks or tasks in a service using the replica scheduling strategy. For tasks in a service using the daemon scheduling strategy, the instances are not protected.
11. Choose **Create** to complete the capacity provider creation.

    Important
    If you receive an error during this step, try logging out and back in to the console. If the error does not clear, we recommend using the AWS CLI instead. For more information, see To create a capacity provider (AWS CLI) (p. 45).

To create a capacity provider (AWS CLI)

Use the following command to create a new capacity provider.

- create-capacity-provider (AWS CLI)

```
aws ecs create-capacity-provider \
     --name CapacityProviderName \
     --auto-scaling-group-provider
 autoScalingGroupArn="AutoScalingGroupARN",managedScaling=\{status='ENABLED|
DISABLED',targetCapacity=integer,minimumScalingStepSize=integer,maximumScalingStepSize=integer\},mana
DISABLED" \
     --region us-east-2
```

If you prefer to use a JSON input file with the `create-capacity-provider` command, use the following command to generate a CLI skeleton.

```
aws ecs create-capacity-provider --generate-cli-skeleton
```

Creating a Cluster

When a new Amazon ECS cluster is created, you specify one or more capacity providers to associate with the cluster. The associated capacity providers determine the infrastructure to run your tasks on.

For AWS Management Console steps, see Creating a Cluster (p. 38).

To create a cluster with a capacity provider (AWS CLI)

Use the following command to create a new capacity provider.

- create-cluster (AWS CLI)

```
aws ecs create-cluster \
     --cluster-name ASGCluster \
     --capacity-providers CapacityProviderA CapacityProviderB \
     --default-capacity-provider-strategy
 capacityProvider=CapacityProviderA,weight=1,base=1
 capacityProvider=CapacityProviderB,weight=1 \
     --region us-west-2
```

If you prefer to use a JSON input file with the `create-cluster` command, use the following command to generate a CLI skeleton.

```
aws ecs create-cluster --generate-cli-skeleton
```

Tutorial: Using Cluster Auto Scaling with the AWS Management Console

Amazon ECS cluster auto scaling can be set up and configured using the AWS Management Console, AWS CLI, or Amazon ECS API.

This tutorial walks you through creating the resources for cluster auto scaling using the AWS Management Console. Where resources require a name, we will use the prefix `ConsoleTutorial` to ensure they all have unique names and to make them easy to locate.

For an AWS CLI tutorial, see Tutorial: Using Cluster Auto Scaling with the AWS CLI (p. 52).

Topics

- Prerequisites (p. 46)
- Step 1: Create an Amazon ECS cluster (p. 46)
- Step 2: Create the Auto Scaling Resources (p. 47)
- Step 3: Create a Capacity Provider (p. 48)
- Step 4: Set a Default Capacity Provider Strategy for the Cluster (p. 49)
- Step 5: Register a Task Definition (p. 49)
- Step 6: Run a Task (p. 50)
- Step 7: Verify (p. 50)
- Step 8: Clean Up (p. 51)

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- 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 Permissions (p. 454) IAM policy example.
- The Amazon ECS container instance IAM role is created. For more information, see Amazon ECS Container Instance IAM Role (p. 486).
- The Amazon ECS service-linked IAM role is created. For more information, see Service-Linked Role for Amazon ECS (p. 473).
- The Auto Scaling service-linked IAM role is created. For more information, see Service-Linked Roles for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

Step 1: Create an Amazon ECS cluster

Use the following steps to create an Amazon ECS cluster. This tutorial uses an empty cluster so that we can manually create the Auto Scaling resources. When you use the AWS Management Console to create a non-empty cluster, Amazon ECS creates an AWS CloudFormation stack along with Auto Scaling resources. We want to avoid creating this AWS CloudFormation stack when using the cluster auto scaling feature.

To create an empty cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. For **Select cluster compatibility**, choose **EC2 Linux + Networking** and then choose **Next step**.
6. For **Cluster name**, enter `ConsoleTutorial-cluster` for the cluster name.
7. Select **Create an empty cluster** and then choose **Create**.

Step 2: Create the Auto Scaling Resources

Use the following steps to create an Auto Scaling launch configuration and Auto Scaling group.

To create an Auto Scaling launch configuration

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. On the navigation pane, under **Auto Scaling**, choose **Launch Configurations**.
4. On the next page, choose **Create launch configuration**.
5. On the **Choose AMI** page, search for and choose the latest Amazon ECS-optimized Amazon Linux 2 AMI in the `us-west-2` Region. The AMI ID can be retrieved using the following link: View AMI ID.
6. On the **Choose Instance Type** page, select `t2.micro`, then choose **Next: Configure details**.
7. On the **Configure details** page, do the following:

   a. For **Name**, enter `ConsoleTutorial-ASGlaunchconfig` for the launch configuration name.

   b. For **IAM role**, select your container instance IAM role. For more information, see Amazon ECS Container Instance IAM Role (p. 486).

   c. Expand the **Advanced Details** section to specify user data for your Amazon ECS container instances.

   Paste the following script into the **User data** field. The `ConsoleTutorial-cluster` cluster was created in the first step.

   ```
   #!/bin/bash
   echo ECS_CLUSTER=ConsoleTutorial-cluster >> /etc/ecs/ecs.config
   ```

   d. Choose **Skip to review**.
8. Choose **Create launch configuration**.

Next, create an Auto Scaling group using that launch configuration.

To create an Auto Scaling group

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. On the navigation pane, under **Auto Scaling**, choose **Launch Configurations**.
4. On the next page, select the launch configuration we created in step 1 and choose **Create Auto Scaling group**.
5. On the **Configure Auto Scaling group details** page, do the following:

a. For **Group name**, enter `ConsoleTutorial-ASG` for the Auto Scaling group name.
b. For **Group size**, enter `0`. The tutorial uses Amazon ECS managed scaling so there is no need to have the Auto Scaling group launch any initial instances.
c. For **Network**, choose a VPC for your Auto Scaling group.
d. For **Subnet**, choose a subnet in your VPC.
e. Expand the **Advanced Details** section. For **Instance Protection**, choose **Protect From Scale In**. This enables you to use managed termination protection for the instances in the Auto Scaling group, which prevents your container instances that contain tasks from being terminated during scale-in actions.

6. Choose **Next: Configure scaling policies**.
7. On the **Configure scaling policies** page, select **Keep this group at its initial size**. The tutorial uses Amazon ECS managed scaling so there is no need to create a scaling policy.
8. Choose **Review**, **Create Auto Scaling group**.
9. Repeat steps 3 to 8 to create a second Auto Scaling group but for **Group name** use `ConsoleTutorial-ASG-burst`.
10. Use the following steps to edit the max capacity value for each of your Auto Scaling groups.
    a. Choose **View your Auto Scaling groups**.
    b. Select your `ConsoleTutorial-ASG` scaling group. From the **Details** tab, choose **Edit**.
    c. For **Max**, enter `100`, then choose **Save**.
11. Repeat step 10 for your `ConsoleTutorial-ASG-burst` scaling group.

Step 3: Create a Capacity Provider

Use the following steps to create an Amazon ECS capacity provider.

To create a capacity provider

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your `ConsoleTutorial-cluster` cluster.
5. On the **Capacity Providers** tab, choose **Create**.
6. On the **Create Capacity Providers** window, do the following.
    a. For **Capacity provider name**, enter `ConsoleTutorial-capacityprovider` for the name.
    b. For **Auto Scaling group**, select the `ConsoleTutorial-ASG` Auto Scaling group you created.
    c. For **Managed scaling**, choose **Enabled**. This enables Amazon ECS to manage the scale-in and scale-out actions for the capacity provider.
    d. For **Target capacity %**, enter `100`.
    e. For **Managed termination protection**, choose **Enabled**. This prevents your container instances that contain tasks and that are in the Auto Scaling group from being terminated during a scale-in action.
    f. Choose **Create**.

       Important
       If you receive an error during this step, try logging out and back in to the console. If the error does not clear, we recommend using the AWS CLI tutorial. For more information, see Tutorial: Using Cluster Auto Scaling with the AWS CLI (p. 52).

    g. Choose **View in cluster** to see your new capacity provider.

h. Repeat steps 4 to 6, creating a second capacity provider with name `ConsoleTutorial-capacityprovider-burst` with your `ConsoleTutorial-ASG-burst` Auto Scaling group.

Step 4: Set a Default Capacity Provider Strategy for the Cluster

When running a task or creating a service, the Amazon ECS console uses the default capacity provider strategy for the cluster. The default capacity provider strategy can be defined by updating the cluster.

To define a default capacity provider strategy

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your `ConsoleTutorial-cluster` cluster.
5. On the **Cluster : ConsoleTutorial-cluster** page, choose **Update Cluster**.
6. For **Default capacity provider strategy** choose, **Add provider**.
7. Select your `ConsoleTutorial-capacityprovider` capacity provider.
8. Choose **Add provider**, select your `ConsoleTutorial-capacityprovider-burst` capacity provider.
9. For **Provider 1**, leave the **Base** value at `0` and leave the **Weight** value at `1`.
10. Choose **Update**. This will add the capacity providers to the default capacity provider strategy for the cluster.
11. Choose **View cluster**.

Step 5: Register a Task Definition

Before you can run a task on your cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses an `amazonlinux` image from Docker Hub and simply sleeps. For more information about the available task definition parameters, see Amazon ECS Task Definitions (p. 73).

To register a task definition

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Task Definitions**, **Create new Task Definition**.
4. On the **Create new Task Definition** page, select **EC2**, **Next step**.
5. Choose **Configure via JSON** and copy and paste the following contents and then choose **Save**, **Create**.

```
{
    "family": "ConsoleTutorial-taskdef",
    "containerDefinitions": [
        {
            "name": "sleep",
            "image": "amazonlinux:2",
            "memory": 20,
            "essential": true,
            "command": [
                "sh",
                "-c",
                "sleep infinity"
            ]
```

```
        }
    ],
    "requiresCompatibilities": [
        "EC2"
    ]
}
```

Step 6: Run a Task

After you have registered a task definition for your account, you can run a task in the cluster. For this tutorial, you run five instances of the `ConsoleTutorial-taskdef` task definition in your `ConsoleTutorial-cluster` cluster.

To run a task

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Task Definitions**.
4. Select your **ConsoleTutorial-taskdef** task definition.
5. From the **Actions** menu, choose **Run Task**.
6. Use the following steps to complete the run task workflow.

    a. For **Capacity provider strategy**, the default capacity provider strategy for the cluster must be selected.
    b. For **Cluster**, select your **ConsoleTutorial-cluster** cluster.
    c. For **Number of tasks**, enter `5`.
    d. For **Placement Templates**, choose **BinPack**.
    e. Choose **Run Task**.

Step 7: Verify

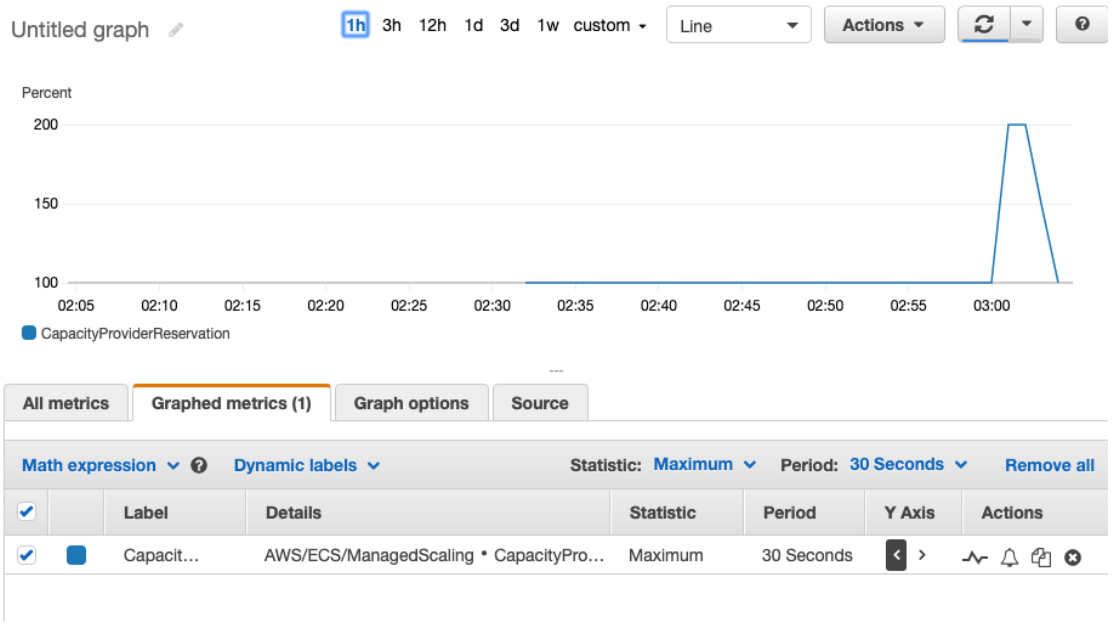
At this point in the tutorial, you should have two Auto Scaling groups with one capacity provider for each of them. The capacity providers have Amazon ECS managed scaling enabled. A cluster was created and five tasks are running.

We can verify that everything is working properly by viewing the CloudWatch metrics, the Auto Scaling group settings, and finally the Amazon ECS cluster task count.

To view the CloudWatch metrics for your cluster

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. On the navigation pane, choose **Metrics**.
4. On the **All metrics** tab, choose **AWS/ECS/ManagedScaling**.
5. Choose **CapacityProviderName, ClusterName**.
6. Choose the metric that corresponds to the **ConsoleTutorial-capacityprovider** capacity provider.
7. On the **Graphed metrics** tab, change **Period** to **30 seconds** and **Statistic** to **Maximum**.

    The value displayed in the graph shows the target capacity value for the capacity provider. It should begin at `100`, which was the target capacity percent we set. You should see it scale up to `200`, which will trigger an alarm for the target tracking scaling policy. The alarm will then trigger the Auto Scaling group to scale out.



8. Steps 5 to 6 can be repeated for your **ConsoleTutorial-capacityprovider-burst** metric.

Use the following steps to view your Auto Scaling group details to confirm that the scale-out action occurred.

To verify the Auto Scaling group scaled out

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
4. For each of your Auto Scaling groups, view the values in the **Instances** and **Desired** columns to confirm your group scaled out to two instances for each group.

Use the following steps to view your Amazon ECS cluster to confirm that the Amazon EC2 instances were registered with the cluster and your tasks transitioned to a `RUNNING` status.

To verify the Auto Scaling group scaled out

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your `ConsoleTutorial-cluster` cluster.
5. On the **ECS Instances** tab, confirm you see four instances registered, which matches your Auto Scaling group values.
6. On the **Tasks** tab, confirm you see five tasks in `RUNNING` status.

Step 8: Clean Up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using. Deleting capacity providers and task definitions are not supported, but there is no cost associated with these resources.

To clean up the tutorial resources

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your `ConsoleTutorial-cluster` cluster.
5. From the **Tasks** tab, choose **Stop All**. Enter the verification and choose **Stop all** again.
6. Delete the Auto Scaling groups using the following steps.
    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
    b. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
    c. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
    d. Select your **ConsoleTutorial-ASG** Auto Scaling group, then from the **Actions** menu choose **Delete.**
    e. Select your **ConsoleTutorial-ASG-burst** Auto Scaling group, then from the **Actions** menu choose **Delete.**
7. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
8. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
9. In the navigation pane, choose **Clusters**.
10. On the **Clusters** page, select your `ConsoleTutorial-cluster` cluster.
11. Choose **Delete Cluster**, enter the confirmation phrase, and choose **Delete**.

Tutorial: Using Cluster Auto Scaling with the AWS CLI

Amazon ECS cluster auto scaling can be set up and configured using the AWS Management Console, AWS CLI, or Amazon ECS API.

This tutorial walks you through creating the resources for cluster auto scaling using the AWS CLI. Where resources require a name, we will use the prefix `CLItutorial` to ensure they all have unique names and to make them easy to locate.

For an AWS Management Console tutorial, see Tutorial: Using Cluster Auto Scaling with the AWS Management Console (p. 46).

Topics

- Prerequisites (p. 52)
- Step 1: Create the Auto Scaling Resources (p. 53)
- Step 2: Create the Amazon ECS Resources (p. 55)
- Step 3: Register a Task Definition (p. 60)
- Step 4: Run a Task (p. 61)
- Step 5: Verify (p. 64)
- Step 6: Clean Up (p. 66)

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 Permissions (p. 454) IAM policy example.
- The Amazon ECS container instance IAM role is created. For more information, see Amazon ECS Container Instance IAM Role (p. 486).
- The Amazon ECS service-linked IAM role is created. For more information, see Service-Linked Role for Amazon ECS (p. 473).
- The Auto Scaling service-linked IAM role is created. For more information, see Service-Linked Roles for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.
- You have a VPC and security group created to use. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

Step 1: Create the Auto Scaling Resources

This step walks you through creating an Auto Scaling launch configuration and two Auto Scaling groups. This step requires that you already have a VPC created along with at least one public subnet and a security group. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

To create the Auto Scaling resources

1. Create an Auto Scaling launch configuration with the following steps. For more information, see Launch Configurations in the *Amazon EC2 Auto Scaling User Guide*.

   a. Create a file named `CLItutorial-launchconfig.json` with the following contents. You must replace the following values:

   - Replace the `ImageId` with the latest Amazon Linux 2 Amazon ECS-optimized AMI. For more information, see Amazon ECS-optimized AMIs (p. 190).
   - Replace the `SecurityGroups` value with your security group ID associated with your VPC.
   - Replace the `IamInstanceProfile` value with the full Amazon Resource Name (ARN) of the instance profile for your Amazon ECS container instance IAM role. An instance profile enables you to pass IAM role information to an Amazon EC2 instance when the instance starts. If your Amazon ECS container instance IAM role is created already, you can retrieve the ARN of the instance profile with the following command. Replace the container instance IAM role name in this example with the name of your container instance IAM role.

```
aws iam list-instance-profiles-for-role --role-name ecsInstanceRole
```

```
{
    "LaunchConfigurationName": "CLItutorial-launchconfig",
    "ImageId": "ami-04240723d51aeeb2d",
    "SecurityGroups": [
        "sg-abcd1234"
    ],
    "InstanceType": "t2.micro",
    "BlockDeviceMappings": [
        {
            "DeviceName": "/dev/xvdcz",
            "Ebs": {
                "VolumeSize": 22,
                "VolumeType": "gp2",
                "DeleteOnTermination": true,
                "Encrypted": true
            }
        }
```

```
    ],
    "InstanceMonitoring": {
        "Enabled": false
    },
    "IamInstanceProfile": "arn:aws:iam::111122223333:instance-
profile/ecsInstanceRole",
    "AssociatePublicIpAddress": true
}
```

b. Create a file named `CLItutorial-userdata.txt` with the following contents. This user data script will be used to register the Amazon EC2 instances created by the Auto Scaling group with the Amazon ECS cluster used in the tutorial, which we have named `CLItutorial-cluster`.

```
#!/bin/bash
echo ECS_CLUSTER=CLItutorial-cluster >> /etc/ecs/ecs.config
```

c. Create the Auto Scaling launch configuration.

```
aws autoscaling create-launch-configuration --cli-input-json file://CLItutorial-
launchconfig.json --user-data file://CLItutorial-userdata.txt --region us-west-2
```

If the command is successful, there will be no output. Use the following command to display the details of your launch configuration.

```
aws autoscaling describe-launch-configurations --launch-configuration-
names CLItutorial-launchconfig --region us-west-2
```

2. Create an Auto Scaling group with the following steps. For more information, see Auto Scaling Groups in the *Amazon EC2 Auto Scaling User Guide*.

a. Create a file named `CLItutorial-asgconfig.json` with the following contents. You must replace the following values:

- Replace the `AvailabilityZones` value with the Availability Zone your subnet exists in.
- Replace the `VPCZoneIdentifier` value with the ID of a subnet in your VPC.
- Replace the `ServiceLinkedRoleARN` value with the full Amazon Resource Name (ARN) of your Auto Scaling service-linked IAM role. For more information, see Service-Linked Roles for Amazon EC2 Auto Scaling in the *Amazon EC2 Auto Scaling User Guide*.

Important
Specifying `true` for the `NewInstancesProtectedFromScaleIn` value is required when using the Amazon ECS managed scaling feature for cluster auto scaling. This tutorial demonstrates using the Amazon ECS managed scaling feature for cluster auto scaling in a later step.

```
{
    "LaunchConfigurationName": "CLItutorial-launchconfig",
    "MinSize": 0,
    "MaxSize": 100,
    "DesiredCapacity": 0,
    "DefaultCooldown": 300,
    "AvailabilityZones": [
        "us-west-2c"
    ],
    "HealthCheckType": "EC2",
    "HealthCheckGracePeriod": 300,
    "VPCZoneIdentifier": "subnet-abcd1234",
    "TerminationPolicies": [
```

```
        "DEFAULT"
    ],
    "NewInstancesProtectedFromScaleIn": true,
    "ServiceLinkedRoleARN": "arn:aws:iam::111122223333:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling"
}
```

b. Create an Auto Scaling group.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name CLItutorial-asg
 --cli-input-json file://CLItutorial-asgconfig.json --region us-west-2
```

If the command is successful, there will be no output.

c. To create a second scaling group, repeat the same command with a different Auto Scaling group name.

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name CLItutorial-
asg-burst --cli-input-json file://CLItutorial-asgconfig.json --region us-west-2
```

If the command is successful, there will be no output.

d. Retrieve the details of the two Auto Scaling groups you just created.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-
names CLItutorial-asg CLItutorial-asg-burst --region us-west-2
```

The output will display the full Amazon Resource Name (ARN) of the two Auto Scaling groups, which you will need for the next step.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "CLItutorial-asg",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:111122223333:autoScalingGroup:24c44d96-606a-427f-826a-
f64ba4cc918c:autoScalingGroupName/CLItutorial-asg",
            "LaunchConfigurationName": "CLItutorial-launchconfig",
            ...
        },
        {
            "AutoScalingGroupName": "CLItutorial-asg-burst",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:111122223333:autoScalingGroup:407c3102-fb00-4a0c-
a1a8-0b242203a262:autoScalingGroupName/CLItutorial-asg-burst",
            "LaunchConfigurationName": "CLItutorial-launchconfig",
            ...
        }
    ]
}
```

Step 2: Create the Amazon ECS Resources

This step will walk you through creating two Amazon ECS capacity providers and one Amazon ECS cluster. You can associate one Auto Scaling group with each capacity provider. This tutorial uses the `us-west-2` Region.

To create the Amazon ECS resources

1. Create an Amazon ECS capacity provider with the following steps.

   a. Create a file named `CLItutorial-capacityprovider.json` with the following contents. Replace the `autoScalingGroupArn` value with the full Amazon Resource Name (ARN) of the first Auto Scaling group you created in step 1.

   ```
   {
       "name": "CLItutorial-capacityprovider",
       "autoScalingGroupProvider": {
           "autoScalingGroupArn": "arn:aws:autoscaling:us-
   west-2:111122223333:autoScalingGroup:24c44d96-606a-427f-826a-
   f64ba4cc918c:autoScalingGroupName/CLItutorial-asg",
           "managedScaling": {
               "status": "ENABLED",
               "targetCapacity": 100,
               "minimumScalingStepSize": 1,
               "maximumScalingStepSize": 100
           },
           "managedTerminationProtection": "ENABLED"
       }
   }
   ```

   b. Create the capacity provider.

   ```
   aws ecs create-capacity-provider --cli-input-json file://CLItutorial-
   capacityprovider.json --region us-west-2
   ```

   The output returns a description of the capacity provider.

   ```
   {
       "capacityProvider": {
           "capacityProviderArn": "arn:aws:ecs:us-west-2:111122223333:capacity-
   provider/CLItutorial-capacityprovider/1a484097-270b-45ea-be85-592924EXAMPLE",
           "name": "CLItutorial-capacityprovider",
           "status": "ACTIVE",
           "autoScalingGroupProvider": {
               "autoScalingGroupArn": "arn:aws:autoscaling:us-
   west-2:111122223333:autoScalingGroup:24c44d96-606a-427f-826a-
   f64ba4cc918c:autoScalingGroupName/CLItutorial-asg",
               "managedScaling": {
                   "status": "ENABLED",
                   "targetCapacity": 100,
                   "minimumScalingStepSize": 1,
                   "maximumScalingStepSize": 100
               },
               "managedTerminationProtection": "ENABLED"
           },
           "tags": []
       }
   }
   ```

2. Create a second Amazon ECS capacity provider with the following steps. The purpose of the second capacity provider will be to provide burst capacity to the cluster. In production you may use Amazon EC2 Spot Instances, but for the purposes of this tutorial we will be using On-Demand Instance.

   a. Create a file named `CLItutorial-capacityprovider-burst.json` with the following contents. Replace the `autoScalingGroupArn` value with the full Amazon Resource Name (ARN) of the second Auto Scaling group you created in step 1.

```
{
    "name": "CLItutorial-capacityprovider-burst",
    "autoScalingGroupProvider": {
        "autoScalingGroupArn": "arn:aws:autoscaling:us-
west-2:111122223333:autoScalingGroup:407c3102-fb00-4a0c-
a1a8-0b242203a262:autoScalingGroupName/CLItutorial-asg-burst",
        "managedScaling": {
            "status": "ENABLED",
            "targetCapacity": 100,
            "minimumScalingStepSize": 1,
            "maximumScalingStepSize": 100
        },
        "managedTerminationProtection": "ENABLED"
    }
}
```

b. Create the capacity provider.

```
aws ecs create-capacity-provider --cli-input-json file://CLItutorial-
capacityprovider-burst.json --region us-west-2
```

The output returns a description of the capacity provider.

```
{
    "capacityProvider": {
        "capacityProviderArn": "arn:aws:ecs:us-west-2:111122223333:capacity-
provider/CLItutorial-capacityprovider-burst/5e4344097-270b-78ea-
be85-592924EXAMPLE",
        "name": "CLItutorial-capacityprovider-burst",
        "status": "ACTIVE",
        "autoScalingGroupProvider": {
            "autoScalingGroupArn": "arn:aws:autoscaling:us-
west-2:111122223333:autoScalingGroup:407c3102-fb00-4a0c-
a1a8-0b242203a262:autoScalingGroupName/CLItutorial-asg-burst",
            "managedScaling": {
                "status": "ENABLED",
                "targetCapacity": 100,
                "minimumScalingStepSize": 1,
                "maximumScalingStepSize": 100
            },
            "managedTerminationProtection": "ENABLED"
        },
        "tags": []
    }
}
```

3. Create an Amazon ECS cluster. The cluster name must match the name you specified in the user data script specified in the Auto Scaling launch configuration created in step 1 of this tutorial. The capacity providers we created in the previous step will be associated with this cluster.

   When a task is run or a service is created, you specify a capacity provider strategy for the tasks to use. Similarly, a default capacity provider strategy can be specified for a cluster. This enables you to run tasks and create services without specifying a capacity provider strategy, as these tasks and actions will use the cluster's default capacity provider strategy. When specifying a default capacity provider strategy, you may optionally specify both a base and weight value. These values are useful when you are associating multiple capacity providers with a cluster. For more information, see Cluster Capacity Provider Concepts (p. 41).

```
aws ecs create-cluster --cluster-name CLItutorial-cluster --capacity-
providers CLItutorial-capacityprovider CLItutorial-capacityprovider-burst --default-
```

```
capacity-provider-strategy capacityProvider=CLItutorial-capacityprovider,weight=1
 capacityProvider=CLItutorial-capacityprovider-burst,weight=1 --region us-west-2
```

The output returns a description of the cluster, including the cluster status and the cluster attachment details. The description, displays the AWS Auto Scaling scaling plans that Amazon ECS creates for you.. A scaling plan is created for each capacity provider.

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-cluster",
        "clusterName": "CLItutorial-cluster",
        "status": "PROVISIONING",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "capacityProviders": [
            "CLItutorial-capacityprovider",
            "CLItutorial-capacityprovider-burst"
        ],
        "defaultCapacityProviderStrategy": [
            {
                "capacityProvider": "CLItutorial-capacityprovider",
                "weight": 1,
                "base": 0
            },
            {
                "capacityProvider": "CLItutorial-capacityprovider-burst",
                "weight": 1,
                "base": 0
            }
        ],
        "attachments": [
            {
                "id": "4aaee2ac-2a66-457c-b0df-a0bc871f5ead",
                "type": "asp",
                "status": "PRECREATED",
                "details": [
                    {
                        "name": "capacityProviderName",
                        "value": "CLItutorial-capacityprovider"
                    },
                    {
                        "name": "scalingPlanName",
                        "value": "ECSManagedAutoScalingPlan-27eb1e2a-5698-4ae7-
b382-1553b8ba1095"
                    }
                ]
            },
            {
                "id": "03e99543-935d-4ea2-9a96-4b9dd63d320f",
                "type": "asp",
                "status": "PRECREATED",
                "details": [
                    {
                        "name": "capacityProviderName",
```

```
                        "value": "CLItutorial-capacityprovider-burst"
                    },
                    {
                        "name": "scalingPlanName",
                        "value": "ECSManagedAutoScalingPlan-f9ea310b-680e-4654-
b8c6-1c4862b29a77"
                    }
                ]
            }
        ],
        "attachmentsStatus": "UPDATE_IN_PROGRESS"
    }
}
```

4. Before you continue to the next step, you must ensure that the cluster is in an `ACTIVE` state, that each of your cluster attachments are in a `CREATED` state, and that the attachment status is in `UPDATE_COMPLETE` state. This can be done by describing the cluster.

```
aws ecs describe-clusters --clusters CLItutorial-cluster --include ATTACHMENTS --region
 us-west-2
```

The output returns a description of your cluster. Verify the cluster and attachment status fields.

```
{
    "cluster": {
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-cluster",
        "clusterName": "CLItutorial-cluster",
        "status": "ACTIVE",
        "registeredContainerInstancesCount": 0,
        "runningTasksCount": 0,
        "pendingTasksCount": 0,
        "activeServicesCount": 0,
        "statistics": [],
        "tags": [],
        "settings": [
            {
                "name": "containerInsights",
                "value": "disabled"
            }
        ],
        "capacityProviders": [
            "CLItutorial-capacityprovider",
            "CLItutorial-capacityprovider-burst"
        ],
        "defaultCapacityProviderStrategy": [
            {
                "capacityProvider": "CLItutorial-capacityprovider",
                "weight": 1,
                "base": 0
            },
            {
                "capacityProvider": "CLItutorial-capacityprovider-burst",
                "weight": 1,
                "base": 0
            }
        ],
        "attachments": [
            {
                "id": "4aaee2ac-2a66-457c-b0df-a0bc871f5ead",
                "type": "asp",
                "status": "CREATED",
                "details": [
                    {
```

```
                        "name": "capacityProviderName",
                        "value": "CLItutorial-capacityprovider"
                    },
                    {
                        "name": "scalingPlanName",
                        "value": "ECSManagedAutoScalingPlan-27eb1e2a-5698-4ae7-
b382-1553b8ba1095"
                    }
                ]
            },
            {
                "id": "03e99543-935d-4ea2-9a96-4b9dd63d320f",
                "type": "asp",
                "status": "CREATED",
                "details": [
                    {
                        "name": "capacityProviderName",
                        "value": "CLItutorial-capacityprovider-burst"
                    },
                    {
                        "name": "scalingPlanName",
                        "value": "ECSManagedAutoScalingPlan-f9ea310b-680e-4654-
b8c6-1c4862b29a77"
                    }
                ]
            }
        ],
        "attachmentsStatus": "UPDATE_COMPLETE"
    }
}
```

Step 3: Register a Task Definition

Before you can run a task on your cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses an `amazonlinux` image from Docker Hub and just sleeps. For more information about the available task definition parameters, see Amazon ECS Task Definitions (p. 73).

To register a task definition

1. Create a file named `CLItutorial-taskdef.json` with the following contents.

```
{
    "family": "CLItutorial-taskdef",
    "containerDefinitions": [
        {
            "name": "sleep",
            "image": "amazonlinux:2",
            "memory": 20,
            "essential": true,
            "command": [
                "sh",
                "-c",
                "sleep infinity"
            ]
        }
    ],
    "requiresCompatibilities": [
        "EC2"
    ]
}
```

2. Register the task definition.

```
aws ecs register-task-definition --cli-input-json file://CLItutorial-taskdef.json --
region us-west-2
```

The output returns a description of the task definition after it completes its registration.

```
{
    "taskDefinition": {
        "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/
CLItutorial-taskdef:1",
        "containerDefinitions": [
            {
                "name": "sleep",
                "image": "amazonlinux:2",
                "cpu": 0,
                "memory": 20,
                "portMappings": [],
                "essential": true,
                "command": [
                    "sh",
                    "-c",
                    "sleep infinity"
                ],
                "environment": [],
                "mountPoints": [],
                "volumesFrom": []
            }
        ],
        "family": "sleep360",
        "revision": 1,
        "volumes": [],
        "status": "ACTIVE",
        "placementConstraints": [],
        "compatibilities": [
            "EC2"
        ],
        "requiresCompatibilities": [
            "EC2"
        ]
    }
}
```

Step 4: Run a Task

After you have registered a task definition for your account, you can run a task in the cluster. For this tutorial, you run five instances of the `CLItutorial-taskdef:1` task definition in your `CLItutorial-cluster` cluster.

To run a task

- Run five instances of the `sleep360:1` task definition you registered in the previous step.

```
aws ecs run-task --cluster CLItutorial-cluster --count 5 --task-definition CLItutorial-
taskdef:1 --region us-west-2
```

The output returns a description of the tasks. Each task will have a capacity provider associated with it.

```
{
    "tasks": [
        {
            "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/CLItutorial-
cluster/12648317756d430e8e320bbc4aEXAMPLE",
            "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-
cluster",
            "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/
CLItutorial-taskdef:1",
            "overrides": {
                "containerOverrides": [],
                "inferenceAcceleratorOverrides": []
            },
            "lastStatus": "PROVISIONING",
            "desiredStatus": "RUNNING",
            "cpu": "0",
            "memory": "20",
            "containers": [],
            "version": 1,
            "createdAt": 1574320187.938,
            "group": "family:CLItutorial-taskdef",
            "launchType": "EC2",
            "capacityProviderName": "CLItutorial-capacityprovider-burst",
            "attachments": [],
            "tags": []
        },
        {
            "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/CLItutorial-cluster/
e7f774f1570b4dddaa08626809EXAMPLE",
            "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-
cluster",
            "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/
CLItutorial-taskdef:1",
            "overrides": {
                "containerOverrides": [],
                "inferenceAcceleratorOverrides": []
            },
            "lastStatus": "PROVISIONING",
            "desiredStatus": "RUNNING",
            "cpu": "0",
            "memory": "20",
            "containers": [],
            "version": 1,
            "createdAt": 1574320187.938,
            "group": "family:CLItutorial-taskdef",
            "launchType": "EC2",
            "capacityProviderName": "CLItutorial-capacityprovider-burst",
            "attachments": [],
            "tags": []
        },
        {
            "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/CLItutorial-cluster/
f0f06980486e43438bc75a2184EXAMPLE",
            "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-
cluster",
            "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/
CLItutorial-taskdef:1",
            "overrides": {
                "containerOverrides": [],
                "inferenceAcceleratorOverrides": []
            },
            "lastStatus": "PROVISIONING",
            "desiredStatus": "RUNNING",
            "cpu": "0",
```

```
            "memory": "20",
            "containers": [],
            "version": 1,
            "createdAt": 1574320187.938,
            "group": "family:CLItutorial-taskdef",
            "launchType": "EC2",
            "capacityProviderName": "CLItutorial-capacityprovider",
            "attachments": [],
            "tags": []
        },
        {
            "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/CLItutorial-
cluster/7e3e0da4e71d4bf9ba8e4371dcEXAMPLE",
            "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-
cluster",
            "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/
CLItutorial-taskdef:1",
            "overrides": {
                "containerOverrides": [],
                "inferenceAcceleratorOverrides": []
            },
            "lastStatus": "PROVISIONING",
            "desiredStatus": "RUNNING",
            "cpu": "0",
            "memory": "20",
            "containers": [],
            "version": 1,
            "createdAt": 1574320187.938,
            "group": "family:CLItutorial-taskdef",
            "launchType": "EC2",
            "capacityProviderName": "CLItutorial-capacityprovider",
            "attachments": [],
            "tags": []
        },
        {
            "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/CLItutorial-cluster/
c71afd510c6e4ae58b86da3490EXAMPLE",
            "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-
cluster",
            "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/
CLItutorial-taskdef:1",
            "overrides": {
                "containerOverrides": [],
                "inferenceAcceleratorOverrides": []
            },
            "lastStatus": "PROVISIONING",
            "desiredStatus": "RUNNING",
            "cpu": "0",
            "memory": "20",
            "containers": [],
            "version": 1,
            "createdAt": 1574320187.938,
            "group": "family:CLItutorial-taskdef",
            "launchType": "EC2",
            "capacityProviderName": "CLItutorial-capacityprovider",
            "attachments": [],
            "tags": []
        }
    ],
    "failures": []
}
```

Step 5: Verify

At this point in the tutorial you should have two Auto Scaling groups with one capacity provider for each of them. The capacity providers have Amazon ECS managed scaling enabled. A cluster was created and five tasks are running. The result should be your `CLItutorial-asg` scaling group should contain two instances, each with two tasks running on them, and your `CLItutorial-asg-burst` scaling group should contain two instances, with a single task running on one of them.

To verify the scaling

1. Describe your cluster to determine how many container instances have been registered to it.

```
aws ecs describe-clusters --clusters CLItutorial-cluster --include ATTACHMENTS --
region us-west-2
```

The output returns a description of the cluster. The following snippet confirms that the correct number of container instances were registered.

```
{
    "clusters": [
        {
            "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/CLItutorial-
cluster",
            "clusterName": "CLItutorial-cluster",
            "status": "ACTIVE",
            "registeredContainerInstancesCount": 3,
            "runningTasksCount": 3,
            ...
            "capacityProviders": [
                "CLItutorial-capacityprovider-burst",
                "CLItutorial-capacityprovider"
            ],
            "defaultCapacityProviderStrategy": [
                {
                    "capacityProvider": "CLItutorial-capacityprovider",
                    "weight": 1,
                    "base": 0
                },
                {
                    "capacityProvider": "CLItutorial-capacityprovider-burst",
                    "weight": 1,
                    "base": 0
                }
            ],
            "attachments": [
                {
                    "id": "09f0a708-a91c-421f-a4a8-85db689a2244",
                    "type": "asp",
                    "status": "CREATED",
                    "details": [
                        {
                            "name": "capacityProviderName",
                            "value": "CLItutorial-capacityprovider-burst"
                        },
                        {
                            "name": "scalingPlanName",
                            "value": "ECSManagedAutoScalingPlan-811242a0-
b26c-418c-86c0-2da93feb23e3"
                        }
                    ]
                },
                {
```

```
                    "id": "b3f407db-3d5b-4b51-88cd-bcd233d70682",
                    "type": "asp",
                    "status": "CREATED",
                    "details": [
                        {
                            "name": "capacityProviderName",
                            "value": "CLItutorial-capacityprovider"
                        },
                        {
                            "name": "scalingPlanName",
                            "value": "ECSManagedAutoScalingPlan-
dcd587aa-484f-45d0-8385-5653da381038"
                        }
                    ]
                }
            ],
            "attachmentsStatus": "UPDATE_COMPLETE"
        }
    ],
    "failures": []
}
```

2. Describe your Auto Scaling groups to verify that the scaling plans set the proper desired capacity values.

```
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names CLItutorial-
asg CLItutorial-asg-burst --region us-west-2
```

The output returns a description of the Auto Scaling groups. The following snippet confirms the desired capacity and container instance details of each Auto Scaling group.

```
{
    "AutoScalingGroups": [
        {
            "AutoScalingGroupName": "CLItutorial-asg-burst",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:111122223333:autoScalingGroup:89e0ae05-4f1d-46b7-
b862-331a3fef4488:autoScalingGroupName/CLItutorial-asg-burst",
            "LaunchConfigurationName": "CLItutorial-launchconfig",
            "MinSize": 0,
            "MaxSize": 10000,
            "DesiredCapacity": 1,
            ...
            "Instances": [
                {
                    "InstanceId": "i-0880a00d7040f2b6a",
                    "AvailabilityZone": "us-west-2c",
                    "LifecycleState": "InService",
                    "HealthStatus": "Healthy",
                    "LaunchConfigurationName": "CLItutorial-launchconfig",
                    "ProtectedFromScaleIn": true
                }
            ],
            ...
            "NewInstancesProtectedFromScaleIn": true,
            "ServiceLinkedRoleARN": "arn:aws:iam::111122223333:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling"
        },
        {
            "AutoScalingGroupName": "CLItutorial-asg",
            "AutoScalingGroupARN": "arn:aws:autoscaling:us-
west-2:111122223333:autoScalingGroup:292c0be4-4218-4f65-9903-
df914d398658:autoScalingGroupName/CLItutorial-asg",
```

```
            "LaunchConfigurationName": "CLItutorial-launchconfig",
            "MinSize": 0,
            "MaxSize": 10000,
            "DesiredCapacity": 2,
            ...
            "Instances": [
                {
                    "InstanceId": "i-0d29b646b6f6b69c5",
                    "AvailabilityZone": "us-west-2c",
                    "LifecycleState": "InService",
                    "HealthStatus": "Healthy",
                    "LaunchConfigurationName": "CLItutorial-launchconfig",
                    "ProtectedFromScaleIn": true
                },
                {
                    "InstanceId": "i-0eb1b33b75fb12da5",
                    "AvailabilityZone": "us-west-2c",
                    "LifecycleState": "InService",
                    "HealthStatus": "Healthy",
                    "LaunchConfigurationName": "CLItutorial-launchconfig",
                    "ProtectedFromScaleIn": true
                }
            ],
            ...
            "NewInstancesProtectedFromScaleIn": true,
            "ServiceLinkedRoleARN": "arn:aws:iam::111122223333:role/aws-service-role/
autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling"
        }
    ]
}
```

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. Deleting capacity providers and task definitions are not supported, but there is no cost associated with these resources.

To clean up the tutorial resources

1. List the tasks in your cluster.

```
aws ecs list-tasks --cluster CLItutorial-cluster --region us-west-2
```

   The output returns a list of the tasks with the full ARNs.

```
{
    "taskArns": [
        "arn:aws:ecs:us-west-2:111122223333:task/3769f4fd-
fe01-4629-9c9d-19b36bEXAMPLE",
        "arn:aws:ecs:us-west-2:111122223333:task/3a311591-
de1a-4d6a-89dd-2be110EXAMPLE",
        "arn:aws:ecs:us-
west-2:111122223333:task/5b46ed48-25c0-4eee-842d-8f89c6EXAMPLE",
        "arn:aws:ecs:us-
west-2:111122223333:task/61b417b4-5a79-4cf7-9cef-18f5d4EXAMPLE",
        "arn:aws:ecs:us-west-2:111122223333:task/6272948e-09e9-4987-a26f-802de9EXAMPLE"
    ]
}
```

2. Stop each of the tasks in your cluster using either the ID or full ARN of the task from the output of the previous step. Repeat this step for each of the five running tasks.

```
aws ecs stop-task --cluster CLItutorial-cluster --task 3769f4fd-
fe01-4629-9c9d-19b36bEXAMPLE --region us-west-2
```

The output returns a description of the task, with an updated desired status of `STOPPED`.

3. Delete the Auto Scaling groups using the following steps. Specifying the `--force-delete` parameter will terminate the container instances as well.

   a. Delete the first Auto Scaling group.

   ```
   aws autoscaling delete-auto-scaling-group --auto-scaling-group-name CLItutorial-asg
    --force-delete --region us-west-2
   ```

   b. Delete the second Auto Scaling group.

   ```
   aws autoscaling delete-auto-scaling-group --auto-scaling-group-name CLItutorial-
   asg-burst --force-delete --region us-west-2
   ```

4. Delete the Amazon ECS cluster.

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

Using AWS Fargate Capacity Providers

Amazon ECS cluster capacity providers enable you to use both Fargate and Fargate Spot capacity with your Amazon ECS tasks. For more information about cluster capacity providers, see Amazon ECS Cluster Capacity Providers (p. 40).

With Fargate Spot you can run interruption tolerant Amazon ECS tasks at a discounted rate compared to the Fargate price. Fargate Spot runs tasks on spare compute capacity. When AWS needs the capacity back, your tasks will be interrupted with a two-minute warning. This is described in further detail below.

Topics

- Fargate Capacity Provider Considerations (p. 67)
- Handling Fargate Spot Termination Notices (p. 68)
- Creating a New Cluster That Uses Fargate Capacity Providers (p. 69)
- Adding Fargate Capacity Providers To An Existing Cluster (p. 69)
- Running Tasks Using a Fargate Capacity Provider (p. 70)

Fargate Capacity Provider Considerations

The following should be considered when using Fargate capacity providers.

- The Fargate and Fargate Spot capacity providers do not need to be created. They are available to all accounts and only need to be associated with a cluster to be available for use.
- When a new cluster is created using the Amazon ECS console along with the **Networking only** cluster template, the `FARGATE` and `FARGATE_SPOT` capacity providers are associated with the new cluster automatically.
- To add the `FARGATE` and `FARGATE_SPOT` capacity providers to an existing cluster, you must use the AWS CLI or API. For more information, see Adding Fargate Capacity Providers To An Existing Cluster (p. 69).

- Using Fargate Spot requires that your task use platform version 1.3.0 or later. For more information, see AWS Fargate Platform Versions (p. 33).
- When tasks using the Fargate and Fargate Spot capacity providers are stopped, a task state change event is sent to Amazon EventBridge. The stopped reason describes the cause. For more information, see Task State Change Events (p. 432).
- A cluster may contain a mix of Fargate and Auto Scaling group capacity providers, however a capacity provider strategy may only contain either Fargate or Auto Scaling group capacity providers, but not both. For more information, see Auto Scaling Group Capacity Providers in the *Amazon Elastic Container Service Developer Guide*.

Handling Fargate Spot Termination Notices

When tasks using Fargate Spot capacity are stopped due to a Spot interruption, a two-minute warning is sent before a task is stopped. The warning is sent as a task state change event to Amazon EventBridge and a SIGTERM signal to the running task. When using Fargate Spot as part of a service, the service scheduler will receive the interruption signal and attempt to launch additional tasks on Fargate Spot if capacity is available.

To ensure that your containers exit gracefully before the task stops, the following can be configured:

- A `stopTimeout` value of `120` seconds or less can be specified in the container definition that the task is using. Specifying a `stopTimeout` value gives you time between the moment the task state change event is received and the point at which the container is forcefully stopped. For more information, see Container Timeouts (p. 108).
- The SIGTERM signal must be received from within the container to perform any cleanup actions.

The following is a snippet of a task state change event displaying the stopped reason and stop code for a Fargate Spot interruption.

```
{
  "version": "0",
  "id": "9bcdac79-b31f-4d3d-9410-fbd727c29fab",
  "detail-type": "ECS Task State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:task/b99d40b3-5176-4f71-9a52-9dbd6f1cebef"
  ],
  "detail": {
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "createdAt": "2016-12-06T16:41:05.702Z",
    "desiredStatus": "STOPPED",
    "lastStatus": "RUNNING",
    "stoppedReason": "Your Spot Task was interrupted.",
    "stopCode": "TerminationNotice",
    "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/
b99d40b3-5176-4f71-9a52-9dbd6fEXAMPLE",
    ...
  }
}
```

The following is an event pattern that is used to create an EventBridge rule for Amazon ECS task state change events. You can optionally specify a cluster in the `detail` field to receive task state change events for. For more information, see Creating an EventBridge Rule in the *Amazon EventBridge User Guide*.

```
{
```

```
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Task State Change"
    ],
    "detail": {
        "clusterArn": [
            "arn:aws:ecs:us-west-2:111122223333:cluster/default"
        ]
    }
}
```

Creating a New Cluster That Uses Fargate Capacity Providers

When a new Amazon ECS cluster is created, you specify one or more capacity providers to associate with the cluster. The associated capacity providers determine the infrastructure to run your tasks on.

When using the AWS Management Console, the `FARGATE` and `FARGATE_SPOT` capacity providers are associated with the cluster automatically when using the **Networking only** cluster template. For more information, see Creating a Cluster (p. 38).

To create an Amazon ECS cluster using Fargate capacity providers (AWS CLI)

Use the following command to create a new cluster and associate both the Fargate and Fargate Spot capacity providers with it.

- create-cluster (AWS CLI)

```
aws ecs create-cluster \
     --cluster-name FargateCluster \
     --capacity-providers FARGATE FARGATE_SPOT \
     --region us-west-2
```

Adding Fargate Capacity Providers To An Existing Cluster

You can update the pool of available capacity providers for an existing Amazon ECS cluster by using the PutClusterCapacityProviders API.

Adding either the Fargate or Fargate Spot capacity providers to an existing cluster is not supported in the AWS Management Console. You must either create a new Fargate cluster in the console or add the Fargate or Fargate Spot capacity providers to the existing cluster using the Amazon ECS API or AWS CLI.

To add the Fargate capacity providers to an existing cluster (AWS CLI)

Use the following command to add the Fargate and Fargate Spot capacity providers to an existing cluster. If the specified cluster has existing capacity providers associated with it, you must specify all existing capacity providers in addition to any new ones you want to add. Any existing capacity

providers associated with a cluster that are omitted from a PutClusterCapacityProviders API call will be disassociated from the cluster. You can only disassociate an existing capacity provider from a cluster if it's not being used by any existing tasks. These same rules apply to the cluster's default capacity provider strategy. If the cluster has an existing default capacity provider strategy defined, it must be included in the PutClusterCapacityProviders API call. Otherwise, it will be overwritten.

- put-cluster-capacity-providers (AWS CLI)

```
aws ecs put-cluster-capacity-providers \
     --cluster FargateCluster \
     --capacity-providers FARGATE
 FARGATE_SPOT existing_capacity_provider1 existing_capacity_provider2 \
     --default-capacity-provider-strategy existing_default_capacity_provider_strategy \
     --region us-west-2
```

Running Tasks Using a Fargate Capacity Provider

You can run a task or create a service using either the Fargate or Fargate Spot capacity providers by specifying a capacity provider strategy. If no capacity provider strategy is provided, the cluster's default capacity provider strategy is used.

Running a task using the Fargate or Fargate Spot capacity providers is supported in the AWS Management Console. You must add the Fargate or Fargate Spot capacity providers to cluster's default capacity provider strategy if using the AWS Management Console. When using the Amazon ECS API or AWS CLI you can specify either a capacity provider strategy or use the cluster's default capacity provider strategy.

To run a task using a Fargate capacity provider (AWS CLI)

Use the following command to run a task using the Fargate and Fargate Spot capacity providers.

- run-task (AWS CLI)

```
aws ecs run-task \
     --capacity-provider-strategy capacityProvider=FARGATE,weight=1
 capacityProvider=FARGATE_SPOT,weight=1 \
     --cluster FargateCluster \
     --task-definition task-def-family:revision \
     --network-configuration
 "awsvpcConfiguration={subnets=[string,string],securityGroups=[string,string],assignPublicIp=string}"
 \
     --count integer \
     --region us-west-2
```

Create a service using a Fargate capacity provider (AWS CLI)

Use the following command to create a service using the Fargate and Fargate Spot capacity providers.

- create-service (AWS CLI)

```
aws ecs create-service \
     --capacity-provider-strategy capacityProvider=FARGATE,weight=1
 capacityProvider=FARGATE_SPOT,weight=1 \
     --cluster FargateCluster \
     --service-name FargateService \
     --task-definition task-def-family:revision \
```

```
    --network-configuration
"awsvpcConfiguration={subnets=[string,string],securityGroups=[string,string],assignPublicIp=string}"
\
    --desired-count integer \
    --region us-west-2
```

Updating Cluster Settings

Cluster settings enable you to configure parameters for your existing Amazon ECS clusters. You can update cluster settings using the Amazon ECS API, AWS CLI or SDKs. Currently, the only supported cluster setting is `containerInsights`, which allows you to enable or disable CloudWatch Container Insights for an existing cluster. To enable CloudWatch Container Insights for a new cluster, that can be done in the AWS Management Console during cluster creation. For more information, see Creating a Cluster (p. 38).

> **Important**
> Currently, if you delete an existing cluster that does not have Container Insights enabled and then create a new cluster with the same name with Container Insights enabled, Container Insights will not actually be enabled. If you want to preserve the same name for your existing cluster and enable Container Insights, you must wait 7 days before you can re-create it.

To update the settings for a cluster using the command line

Use one of the following commands to update the setting for a cluster.

- update-cluster-settings (AWS CLI)

```
aws ecs update-cluster-settings --cluster cluster_name_or_arn --settings
 name=containerInsights,value=enabled|disabled --region us-east-1
```

Deleting a Cluster

If you are finished using a cluster, you can delete it. Once deleted, the cluster will transition to the `INACTIVE` state. Clusters with an `INACTIVE` status may remain discoverable in your account for a period of time. However, this behavior is subject to change in the future, so you should not rely on `INACTIVE` clusters persisting.

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. 72) of the following procedure changes based on that condition.

If your cluster was created with the AWS Management Console then the AWS CloudFormation stack that was created for your cluster is also deleted when you delete your cluster. If you have added or modified the underlying cluster resources you may receive an error when attempting to delete the cluster. AWS CloudFormation refers to this as *stack drift*. For more information on detecting drift on an existing AWS CloudFormation stack, see Detect Drift on an Entire CloudFormation Stack in the *AWS CloudFormation User Guide*.

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. 248).

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

   - **Deleting the cluster also deletes the AWS CloudFormation stack EC2ContainerService-*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 AWS 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. 248). You can visit the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation/ to update or delete any of these resources.

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. 84).

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. 73).

Topics

- Application Architecture (p. 73)
- Creating a Task Definition (p. 75)
- Task definition parameters (p. 84)
- Amazon ECS Launch Types (p. 119)
- Working with GPUs on Amazon ECS (p. 121)
- Using Data Volumes in Tasks (p. 124)
- Task Networking with the awsvpc Network Mode (p. 141)
- Using the awslogs Log Driver (p. 143)
- Custom Log Routing (p. 150)
- Private Registry Authentication for Tasks (p. 160)
- Specifying Sensitive Data (p. 163)
- Specifying environment variables (p. 175)
- Example Task Definitions (p. 177)
- Updating a Task Definition (p. 181)
- Deregistering Task Definition Revisions (p. 182)

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.

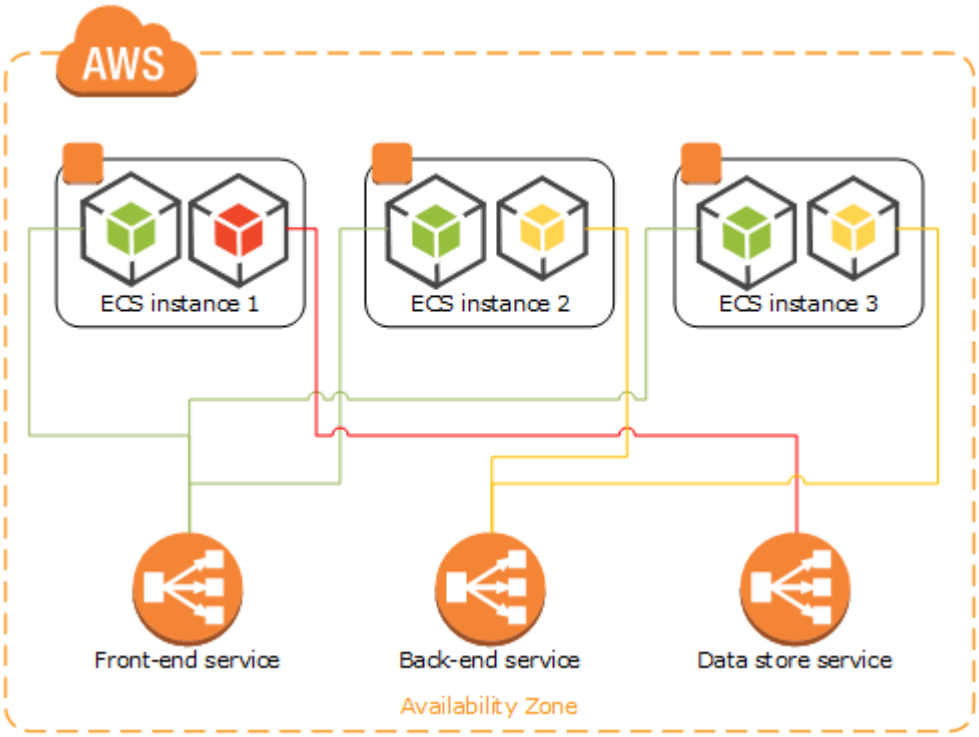
As an example, imagine an application that consists of the following components:

- 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. 352). In your services, you can associate containers with Elastic Load Balancing load balancers. For more information, see Service Load Balancing (p. 376). 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. 363).

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. 84).

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**.

   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 API operations on your behalf. For more information, see IAM Roles for Tasks (p. 489).

   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. 491).
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. 482).
5. For **Task size**, choose a value for **Task memory (GB)** and **Task CPU (vCPU)**. The table below shows the valid combinations.

CPU value	Memory value
256 (.25 vCPU)	512 MB, 1 GB, 2 GB
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments

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. 84).

   c. Choose **Add** to add your container to the task definition.
7. (Optional) For **Service Integration**, to configure the parameters for App Mesh integration choose **Enable App Mesh integration** and then do the following:

   a. For **Application container name**, choose the container name to use for the App Mesh application. This container must already be defined within the task definition.

b. For **Envoy image**, enter 840364872350.dkr.ecr.*us-west-2*.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod.

c. For **Mesh name**, choose the App Mesh service mesh to use. This must already be created in order for it to show up. For more information, see Service Meshes in the *AWS App Mesh User Guide*.

d. For **Virtual node name**, choose the App Mesh virtual node to use. This must already be created in order for it to show up. For more information, see Virtual Nodes in the *AWS App Mesh User Guide*.

e. For **Virtual node port**, this will be pre-populated with the listener port set on the virtual node.

f. Choose **Apply**, **Confirm**. This will create a new Envoy proxy container to the task definition, as well as the settings to support it. It will then pre-populate the App Mesh proxy configuration settings for the next step.

8. (Optional) For **Proxy Configuration**, verify all of the pre-populated values. For more information on these fields, see the JSON tab in Update Services.

9. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:

   a. For **Type**, choose the log router type to use.

   b. For **Image**, type the image URI for your log router container. If you chose the `fluentbit` log router type, the **Image** field prepopulates with the AWS for Fluent Bit image. For more information, see Using the AWS for Fluent Bit Image (p. 152).

   c. Choose **Apply**. This creates a new log router container to the task definition named `log_router`, and applies the settings to support it. If you make changes to the log router integration fields, choose **Apply** again to update the FireLens container.

10. (Optional) To define data volumes for your task, choose **Add volume**. For more information, see Using Data Volumes in Tasks (p. 124).

    - For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

11. 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.

12. 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. 489).

   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. 486).

   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. 491).

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. If you select **Enable App Mesh integration** in a following step, then you must select `awsvpc`.

   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, seeAmazon ECS Task Execution IAM Role (p. 482).

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. 84).

   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. 323).

9. (Optional) For **Service Integration**, to configure the parameters for App Mesh integration choose **Enable App Mesh integration** and then do the following:

   a. For **Application container name**, choose the container name to use for the App Mesh application. This container must already be defined within the task definition.

b. For **Envoy image**, enter 840364872350.dkr.ecr.*us-west-2*.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod.

c. For **Mesh name**, choose the App Mesh service mesh to use. This must already be created in order for it to show up. For more information, see Service Meshes in the *AWS App Mesh User Guide*.

d. For **Virtual node name**, choose the App Mesh virtual node to use. This must already be created in order for it to show up. For more information, see Virtual Nodes in the *AWS App Mesh User Guide*.

e. For **Virtual node port**, this will be pre-populated with the listener port set on the virtual node.

f. Choose **Apply**, **Confirm**. This will create a new Envoy proxy container to the task definition, as well as the settings to support it. It will then pre-populate the App Mesh proxy configuration settings for the next step.

10. (Optional) For **Proxy Configuration**, verify all of the pre-populated values. For more information on these fields, see the JSON tab in Update Services.

11. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:

    a. For **Type**, choose the log router type to use.

    b. For **Image**, type the image URI for your log router container. If you chose the `fluentbit` log router type, the **Image** field prepopulates with the AWS for Fluent Bit image. For more information, see Using the AWS for Fluent Bit Image (p. 152).

    c. Choose **Apply**. This creates a new log router container to the task definition named `log_router`, and applies the settings to support it. If you make changes to the log router integration fields, choose **Apply** again to update the FireLens container.

12. (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. 124).

    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.

13. 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.

14. 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. 84).

```
{
    "family": "",
    "taskRoleArn": "",
    "executionRoleArn": "",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            "repositoryCredentials": {
                "credentialsParameter": ""
            },
            "cpu": 0,
            "memory": 0,
            "memoryReservation": 0,
            "links": [
                ""
            ],
            "portMappings": [
                {
                    "containerPort": 0,
                    "hostPort": 0,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                ""
            ],
            "command": [
                ""
            ],
            "environment": [
                {
                    "name": "",
                    "value": ""
                }
            ],
            "environmentFiles": [
                {
                    "value": "",
                    "type": "s3"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "",
                    "containerPath": "",
                    "readOnly": true
                }
            ],
            "volumesFrom": [
                {
                    "sourceContainer": "",
                    "readOnly": true
                }
            ],
```

```
            "linuxParameters": {
                "capabilities": {
                    "add": [
                        ""
                    ],
                    "drop": [
                        ""
                    ]
                },
                "devices": [
                    {
                        "hostPath": "",
                        "containerPath": "",
                        "permissions": [
                            "read"
                        ]
                    }
                ],
                "initProcessEnabled": true,
                "sharedMemorySize": 0,
                "tmpfs": [
                    {
                        "containerPath": "",
                        "size": 0,
                        "mountOptions": [
                            ""
                        ]
                    }
                ],
                "maxSwap": 0,
                "swappiness": 0
            },
            "secrets": [
                {
                    "name": "",
                    "valueFrom": ""
                }
            ],
            "dependsOn": [
                {
                    "containerName": "",
                    "condition": "HEALTHY"
                }
            ],
            "startTimeout": 0,
            "stopTimeout": 0,
            "hostname": "",
            "user": "",
            "workingDirectory": "",
            "disableNetworking": true,
            "privileged": true,
            "readonlyRootFilesystem": true,
            "dnsServers": [
                ""
            ],
            "dnsSearchDomains": [
                ""
            ],
            "extraHosts": [
                {
                    "hostname": "",
                    "ipAddress": ""
                }
            ],
            "dockerSecurityOptions": [
                ""
```

```
            ],
            "interactive": true,
            "pseudoTerminal": true,
            "dockerLabels": {
                "KeyName": ""
            },
            "ulimits": [
                {
                    "name": "msgqueue",
                    "softLimit": 0,
                    "hardLimit": 0
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "KeyName": ""
                },
                "secretOptions": [
                    {
                        "name": "",
                        "valueFrom": ""
                    }
                ]
            },
            "healthCheck": {
                "command": [
                    ""
                ],
                "interval": 0,
                "timeout": 0,
                "retries": 0,
                "startPeriod": 0
            },
            "systemControls": [
                {
                    "namespace": "",
                    "value": ""
                }
            ],
            "resourceRequirements": [
                {
                    "value": "",
                    "type": "GPU"
                }
            ],
            "firelensConfiguration": {
                "type": "fluentd",
                "options": {
                    "KeyName": ""
                }
            }
        }
    ],
    "volumes": [
        {
            "name": "",
            "host": {
                "sourcePath": ""
            },
            "dockerVolumeConfiguration": {
                "scope": "task",
                "autoprovision": true,
                "driver": "",
                "driverOpts": {
                    "KeyName": ""
```

```
                },
                "labels": {
                    "KeyName": ""
                }
            },
            "efsVolumeConfiguration": {
                "fileSystemId": "",
                "rootDirectory": "",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": 0,
                "authorizationConfig": {
                    "accessPointId": "",
                    "iam": "ENABLED"
                }
            }
        }
    ],
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": ""
        }
    ],
    "requiresCompatibilities": [
        "EC2"
    ],
    "cpu": "",
    "memory": "",
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "pidMode": "task",
    "ipcMode": "none",
    "proxyConfiguration": {
        "type": "APPMESH",
        "containerName": "",
        "properties": [
            {
                "name": "",
                "value": ""
            }
        ]
    },
    "inferenceAccelerators": [
        {
            "deviceName": "",
            "deviceType": ""
        }
    ]
}
```

You can generate this task definition template using the following AWS CLI command:

```
aws ecs register-task-definition --generate-cli-skeleton
```

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 and container definitions are required in a task definition, while task role, network mode, volumes, task placement constraints, and launch type are optional.

The following are more detailed descriptions for each task definition parameter.

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. 489).

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. 721).

Task execution role

`executionRoleArn`

Type: string

Required: no

The Amazon Resource Name (ARN) of the task execution role that grants the Amazon ECS container agent permission to make AWS API calls on your behalf. The task execution IAM role is required depending on the requirements of your task. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

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. 141). 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. 85)
- Advanced Container Definition Parameters (p. 89)
- Other Container Definition Parameters (p. 103)

Standard Container Definition Parameters

The following task definition parameters are either required or used in most container definitions.

Topics

- Name (p. 86)
- Image (p. 86)
- Memory (p. 86)
- Port Mappings (p. 87)

Name

`name`

Type: string

Required: yes

The name of a container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. If you 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

`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. 160).
- Images in Amazon ECR repositories can be specified by using either the full `registry/repository:tag` or `registry/repository@digest` naming convention. For example, *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`:`*`latest`* or *`aws_account_id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`my-web-app`*`@`*`sha256:94afd1f2e64d908bc90dbca0035a5b567EXAMPLE`*
- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

Memory

`memory`

Type: integer

Required: no

The amount (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed. The total amount of memory reserved for all containers within a task must be lower than the task `memory` value, if one is specified. This parameter maps to `Memory` in the Create a container section of the Docker Remote API and the `--memory` option to **docker run**.

If using the Fargate launch type, this parameter is optional.

If using the EC2 launch type, you must specify either a task-level memory value or a container-level memory value. If you specify both a container-level `memory` and `memoryReservation` value, `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. 240).

`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**.

If a task-level memory value is not specified, you must specify a non-zero integer for one or both of `memory` or `memoryReservation` in a container definition. If you specify both, `memory` must be greater than `memoryReservation`. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance 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.

Port Mappings

`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 Docker version 1.6.0 and later is listed on the instance under `/proc/sys/net/ipv4/ip_local_port_range`. If this kernel parameter is unavailable, the default empheral port range from `49153–65535` 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. 264).

If you are specifying a host port, use the following syntax:

```
"portMappings": [
    {
        "containerPort": integer,
        "hostPort": integer
    }
    ...
]
```

If you want an automatically assigned host port, use the following syntax:

```
"portMappings": [
    {
        "containerPort": integer
    }
    ...
]
```

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. 90)
- Environment (p. 91)
- Network Settings (p. 95)
- Storage and Logging (p. 97)
- Security (p. 101)

- Resource Limits (p. 102)
- Docker Labels (p. 103)

Health Check

`healthCheck`

The container 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.

You can view the health status of both individual containers and a task with the DescribeTasks API operation or when viewing the task details in the console.

The following describes the possible `healthStatus` values for a container:

- `HEALTHY`—The container health check has passed successfully.
- `UNHEALTHY`—The container health check has failed.
- `UNKNOWN`—The container health check is being evaluated or there is no container health check defined.

The following describes the possible `healthStatus` values for a task. The container health check status of nonessential containers do not have an effect on the health status of a task.

- `HEALTHY`—All essential containers within the task have passed their health checks.
- `UNHEALTHY`—One or more essential containers have failed their health check.
- `UNKNOWN`—The essential containers within the task are still having their health checks evaluated or there are no container health checks defined.

If a task is run manually, and not as part of a service, the task will continue its lifecycle regardless of its health status. For tasks that are part of a service, if the task reports as unhealthy then the task will be stopped and the service scheduler will replace it.

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. 264).
- 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. 33).
- 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:

```
CMD-SHELL, curl -f http://localhost/ || exit 1
```

When registering a task definition using the AWS Management Console JSON panel, the AWS CLI, or the APIs, 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. 121).

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. 73).

```
"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"
```

`environmentFiles`

Type: object array

Required: no

A list of files containing the environment variables to pass to a container. This parameter maps to the `--env-file` option to **docker run**.

You can specify up to ten environment files. The file must have a `.env` file extension. Each line in an environment file should contain an environment variable in `VARIABLE=VALUE` format. Lines beginning with `#` are treated as comments and are ignored. For more information on the environment variable file syntax, see Declare default environment variables in file.

If there are individual environment variables specified in the container definition, they take precedence over the variables contained within an environment file. If multiple environment files are specified that contain the same variable, they are processed from the top down. It is recommended to use unique variable names. For more information, see Specifying environment variables (p. 175).

This field is not valid for containers in tasks using the Fargate launch type.

`value`

Type: String

Required: Yes

The Amazon Resource Name (ARN) of the Amazon S3 object containing the environment variable file.

`type`

Type: String

Required: Yes

The file type to use. The only supported value is `s3`.

`environment`

Type: object array

Required: no

The environment variables to pass to a container. This parameter maps to `Env` in the 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" : [
    { "name" : "string", "value" : "string" },
    { "name" : "string", "value" : "string" }
]
```

`secrets`

Type: Object array

Required: No

An object representing the secret to expose to your container. For more information, see Specifying Sensitive Data (p. 163).

`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 the same container instance may be able to communicate with each other without requiring links or host port mappings. The network isolation on a container instance is controlled by security groups and VPC settings.

```
"links": ["name:internalName", ...]
```

`hostname`

Type: string

Required: no

The hostname to use for your container. This parameter maps to `Hostname` in the 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 Array

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 example task definitions using a log configuration, see Example Task Definitions (p. 177).

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. 270).
- 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": "awslogs","fluentd","gelf","json-
file","journald","logentries","splunk","syslog","awsfirelens",
      "options": {"string": "string"
        ...},
 "secretOptions": [{
  "name": "string",
  "valueFrom": "string"
 }]
}
```

`logDriver`

Type: string

Valid values: `"awslogs","fluentd","gelf","json-file","journald","logentries","splunk","syslog","awsfirelens`

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`, `splunk`, and `awsfirelens`.

For tasks using the EC2 launch type, the supported log drivers are `awslogs`, `fluentd`, `gelf`, `json-file`, `journald`, `logentries`,`syslog`, `splunk`, and `awsfirelens`.

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. 143).

For more information about using the `awsfirelens` log driver, see Custom Log Routing.

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. 163).

`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 field is not valid for containers in tasks using the Fargate launch type.

With Windows containers, this parameter can be used to reference a credential spec file when configuring a container for Active Directory authentication. For more information, see Using gMSAs for Windows Containers (p. 723).

This parameter maps to `SecurityOpt` in the Create a container section of the Docker Remote API and the `--security-opt` option to **docker run**.

```
"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. 270).

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**.

Fargate tasks use the default resource limit values with the exception of the `nofile` resource limit parameter which Fargate overrides. The `nofile` resource limit sets a restriction on the number of open files that a container can use. The default `nofile` soft limit is `1024` and hard limit is `4096` for Fargate tasks. These limits can be adjusted in a task definition if your tasks needs to handle a larger number of files. For more information, see Task Resource Limits (p. 29).

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"|"r
        "softLimit": integer,
        "hardLimit": integer
      }
      ...
    ]
```

`name`

Type: string

Valid values: `"core" | "cpu" | "data" | "fsize" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"`

Required: yes, when `ulimits` are used

The `type` of the `ulimit`.

`hardLimit`

Type: integer

Required: yes, when `ulimits` are used

The hard limit for the `ulimit` type.

`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. 75).

Topics

- Linux Parameters (p. 103)
- Container Dependency (p. 107)
- Container Timeouts (p. 108)
- System Controls (p. 109)
- Interactive (p. 110)
- Pseudo Terminal (p. 110)

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
For tasks that use the Fargate launch type, `capabilities` is supported for all platform versions but the `add` parameter is only supported if using platform version 1.4.0 or later.

`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 and platform version 1.4.0, the only supported `add` parameter value is `SYS_PTRACE`.

`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.

`maxSwap`

The total amount of swap memory (in MiB) a container can use. This parameter will be translated to the `--memory-swap` option to docker run where the value would be the sum of the container memory plus the `maxSwap` value.

If a `maxSwap` value of `0` is specified, the container will not use swap. Accepted values are `0` or any positive integer. If the `maxSwap` parameter is omitted, the container will use the swap configuration for the container instance it is running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

Note
If you are using tasks that use the Fargate launch type, the `maxSwap` parameter is not supported.

`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

`swappiness`

This allows you to tune a container's memory swappiness behavior. A `swappiness` value of `0` will cause swapping to not happen unless absolutely necessary. A `swappiness` value of `100` will cause pages to be swapped very aggressively. Accepted values are whole numbers between `0` and `100`. If the `swappiness` parameter is not specified, a default value of `60` is used. If a value is not specified for `maxSwap` then this parameter is ignored. This parameter maps to the `--memory-swappiness` option to docker run.

Note
If you are using tasks that use the Fargate launch type, the `swappiness` parameter is not supported.

`tmpfs`

The container path, mount options, and maximum size (in MiB) of the tmpfs mount. This parameter maps to the `--tmpfs` option to docker run.

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" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" | "private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" | "relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode" | "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"`

`size`

The maximum size (in MiB) of the tmpfs volume.

Type: Integer

Required: Yes

Container Dependency

`dependsOn`

Type: Array of ContainerDependency objects

Required: no

The dependencies defined for container startup and shutdown. A container can contain multiple dependencies. When a dependency is defined for container startup, for container shutdown it is reversed. For an example, see Example: Container Dependency (p. 180).

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. 264). 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. 190).

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. This condition cannot be set on an essential container.
- `SUCCESS` – This condition is the same as `COMPLETE`, but it also requires that the container exits with a `zero` status. This condition cannot be set on an essential container.

- `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

Example values: `120`

Time duration (in seconds) to wait before giving up on resolving dependencies for a container. For example, you specify two containers in a task definition with containerA having a dependency on containerB reaching a `COMPLETE`, `SUCCESS`, or `HEALTHY` status. If a `startTimeout` value is specified for containerB and it 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 Fargate launch type, this parameter requires that the task or service uses platform version 1.3.0 or later. If this parameter is not specified, the default value of 3 minutes is used.

For tasks using the EC2 launch type, if the `startTimeout` parameter is not specified, the value set for the Amazon ECS container agent configuration variable `ECS_CONTAINER_START_TIMEOUT` is used by default. If neither the `startTimeout` parameter or the `ECS_CONTAINER_START_TIMEOUT` agent configuration variable are set, then the default values of 3 minutes for Linux containers and 8 minutes on Windows containers are used. Your 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. 264). 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. 190).

`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 task or service requires platform version 1.3.0 or later. The max stop timeout value is 120 seconds and if the parameter is not specified, the default value of 30 seconds is used.

For tasks using the EC2 launch type, if the `stopTimeout` parameter is not specified, the value set for the Amazon ECS container agent configuration variable `ECS_CONTAINER_STOP_TIMEOUT` is used by default. If neither the `stopTimeout` parameter or the `ECS_CONTAINER_STOP_TIMEOUT` agent configuration variable are set, then the default values of 30 seconds for Linux containers and 30 seconds on Windows containers are used. Container instances require at least version 1.26.0 of the container agent to enable a container stop timeout value. However, we recommend using the

latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 264). 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. 190).

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. 118).

- 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.shmmax" | "kernel.shmmni" | "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 specifed 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

Required: no

When this parameter is `true`, a TTY is allocated. This parameter maps to `Tty` in the Create a container section of the Docker Remote API and the `--tty` option to **docker run**.

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. 124).

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`.

`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`.

`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.

`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 destroyed when the task is cleaned up. Docker volumes that are scoped as `shared` persist after the task stops.

`autoprovision`

Type: Boolean

Default value: `false`

Required: No

If this value is `true`, the Docker volume is created if it does not already exist. This field is only used if the `scope` is `shared`. If the `scope` is `task` then this parameter must either be omitted or set to `false`.

`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`.

`efsVolumeConfiguration`

Type: Object

Required: No

This parameter is specified when using Amazon EFS volumes.

`fileSystemId`

Type: String

Required: Yes

The Amazon EFS file system ID to use.

`rootDirectory`

Type: String

Required: No

The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume will be used. Specifying `/` will have the same effect as omitting this parameter.

`transitEncryption`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether or not to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. Transit encryption must be enabled if Amazon EFS IAM authorization is used. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see Encrypting Data in Transit in the *Amazon Elastic File System User Guide*.

`transitEncryptionPort`

Type: Integer

Required: No

The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you do not specify a transit encryption port, it will use the port selection strategy that the Amazon EFS mount helper uses. For more information, see EFS Mount Helper in the *Amazon Elastic File System User Guide*.

`authorizationConfig`

Type: Object

Required: No

The authorization configuration details for the Amazon EFS file system.

`accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value will be relative to the directory set for the access point. If specified, transit encryption must be enabled in the `EFSVolumeConfiguration`. For more information, see Working with Amazon EFS Access Points in the *Amazon Elastic File System User Guide*.

`iam`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether or not to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the `EFSVolumeConfiguration`. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see IAM Roles for Tasks.

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. 323).

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. 327).

`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. 119).

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. 119).

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 hard limit of CPU units to present for 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. If your cluster does not have any registered container instances with the requested CPU units available, the task will fail. 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:

CPU value	Memory value (MiB)
256 (.25 vCPU)	512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)
512 (.5 vCPU)	1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)
1024 (1 vCPU)	2048 (2 GB), 3072 (3 GB), 4096 (4 GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)
2048 (2 vCPU)	Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)
4096 (4 vCPU)	Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)

`memory`

Type: string

Required: no

Note
This parameter is not supported for Windows containers.

The hard limit of memory (in MiB) to present to the task. It can be expressed as an integer using MiB, for example `1024`, or as a string using GB, for example `1GB` or `1 GB`, in a task definition. When the task definition is registered, a GB value is converted to an integer indicating the MiB.

If using the EC2 launch type, this field is optional and any value can be used. If a task-level memory value is specified then the container-level memory value is optional. If your cluster does not have any registered container instances with the requested memory available, the task will fail. 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. 240).

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 `cpu` parameter:

Memory value (MiB)	CPU value
512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)	256 (.25 vCPU)
1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)	512 (.5 vCPU)
2048 (2 GB), 3072 (3 GB), 4096 (4GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)	1024 (1 vCPU)
Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)	2048 (2 vCPU)
Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)	4096 (4 vCPU)

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. 190).

For tasks using the Fargate launch type, this feature requires that the task or service uses platform version 1.3.0 or later.

Note
This parameter is not supported for Windows containers.

```
"proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "string",
    "properties": [
        {
           "name": "string",
           "value": "string"
        }
    ]
}
```

`type`

Type: String

Value values: `APPMESH`

Required: No

The proxy type. The only supported value is `APPMESH`.

`containerName`

Type: String

Required: Yes

The name of the container that 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.

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. 75).

Topics

- IPC mode (p. 118)
- PID mode (p. 118)

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. 109).

- 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 process namespace with the host Amazon EC2 instance. If `task` is specified, all containers within the specified task share the same process namespace. If no value is specified, the default is a private namespace. For 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.

Amazon ECS Launch Types

An Amazon ECS launch type determines the type of infrastructure on which your tasks and services are hosted.

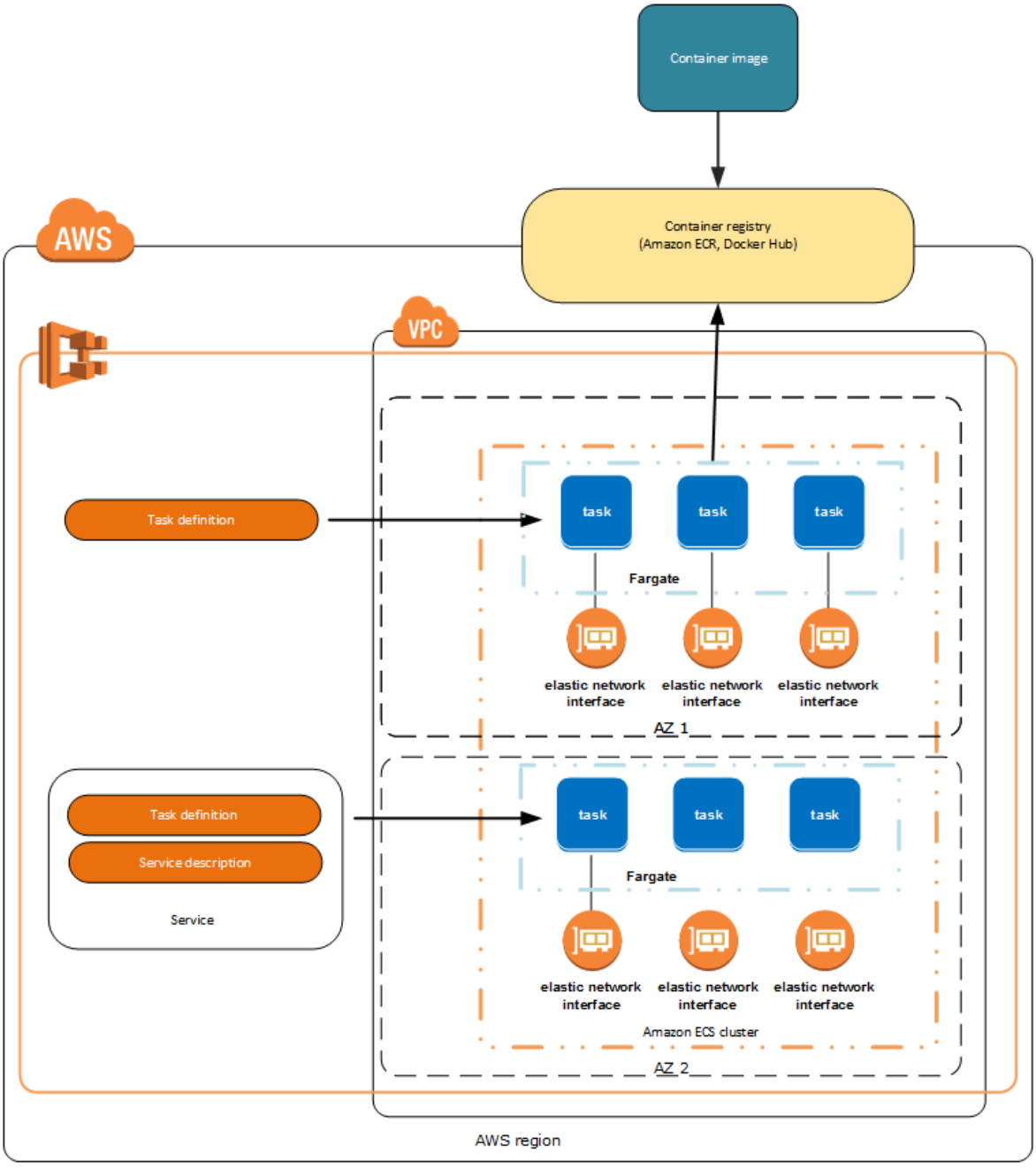
Fargate Launch Type

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:

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1 (`aps1-az1` & `aps1-az3` only)
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
Middle East (Bahrain)	me-south-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

This diagram shows the general architecture:

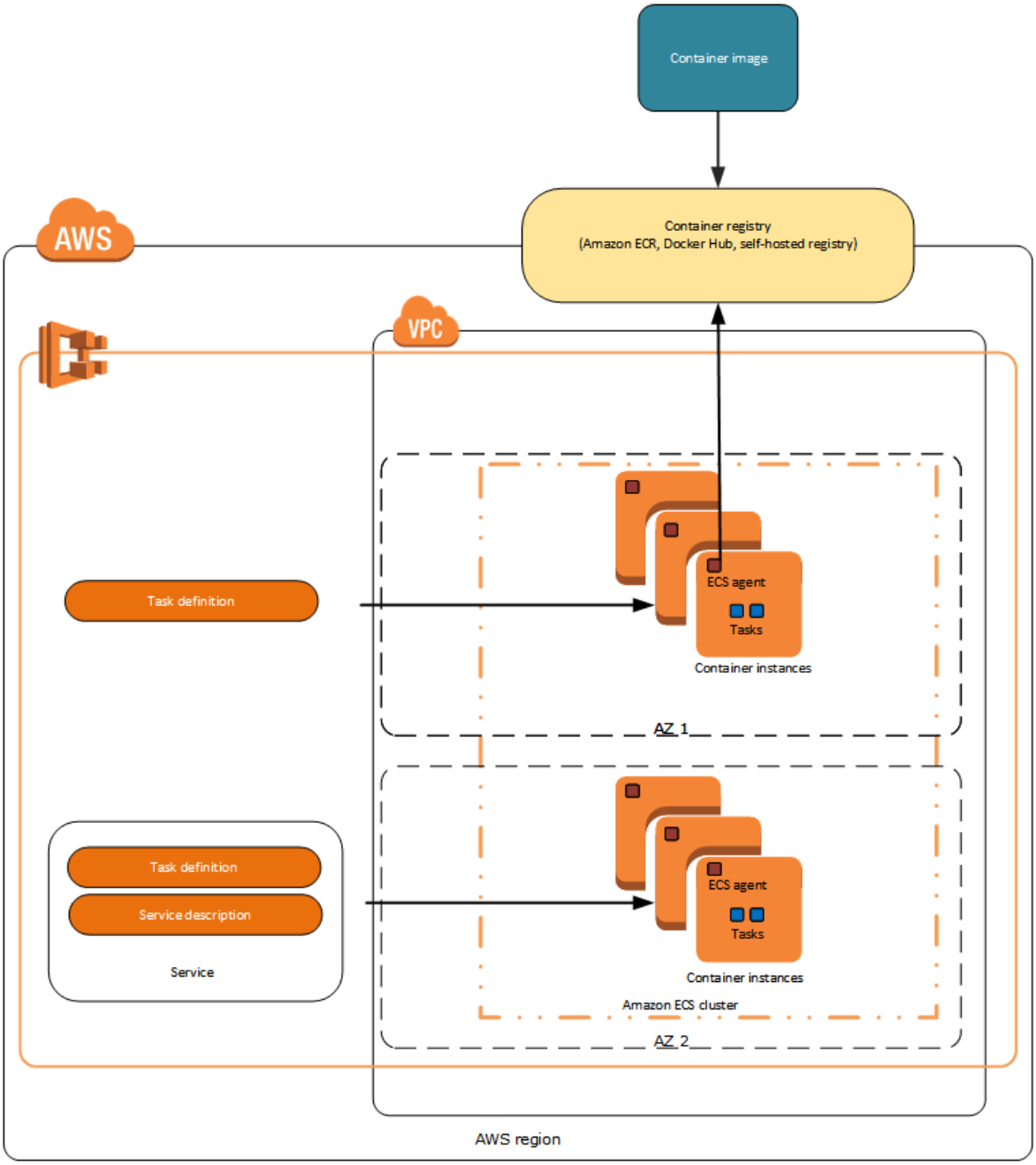


For more information about Amazon ECS with AWS Fargate, see Amazon ECS on AWS Fargate (p. 27).

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:



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, p3, g3, and g4 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. 190).

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, Amazon EC2 P3 Instances, Amazon EC2 G3 Instances, and Amazon EC2 G4 Instances.

Important
The g4 instance type family is supported on version `20190913` and later of the Amazon ECS GPU-optimized AMI. For more information, see Amazon ECS GPU-optimized AMI Versions (p. 200). It is currently not supported in the Create Cluster workflow in the Amazon ECS console. To use these instance types, you must either use the Amazon EC2 console, AWS CLI, or API and manually register the instances to your cluster.

Instance type	GPUs	GPU Memory (GiB)	vCPUs	Memory (GiB)
p2.xlarge	1	12	4	61
p2.8xlarge	8	96	32	488
p2.16xlarge	16	192	64	732
p3.2xlarge	1	16	8	61
p3.8xlarge	4	64	32	244
p3.16xlarge	8	128	64	488
p3dn.24xlarge	8	256	96	768
g3s.xlarge	1	8	4	30.5
g3.4xlarge	1	8	16	122
g3.8xlarge	2	16	32	244
g3.16xlarge	4	32	64	488
g4dn.xlarge	1	16	4	16
g4dn.2xlarge	1	16	8	32
g4dn.4xlarge	1	16	16	64
g4dn.8xlarge	1	16	32	128
g4dn.12xlarge	4	64	48	192
g4dn.16xlarge	1	16	64	256

Topics

- Considerations for Working with GPUs (p. 122)
- Specifying GPUs in Your Task Definition (p. 123)

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. 320).

  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 \
     --placement-constraints type=memberOf,expression="attribute:ecs.instance-type ==
 p2.xlarge" --region us-east-2
```

- For each container that has a GPU resource requirement specified in the container definition, Amazon ECS sets the container runtime to be the NVIDIA container runtime.
- The NVIDIA container runtime requires some environment variables to be set in the container in order to work. For a list of these environment variables, see nvidia-container-runtime. Amazon ECS sets the `NVIDIA_VISIBLE_DEVICES` environment variable value to be a list of the GPU device IDs that Amazon ECS assigns to the container. For the other required environment variables, Amazon ECS does not set them and you should ensure that your container image sets them or they should be set in the container definition.

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. 75).

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
    }
  ],
  "family": "example-ecs-gpu"
}
```

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.

For Fargate tasks, the following data volume formats are supported:

- Amazon EFS volumes for persistent storage. For more information, see Amazon EFS Volumes (p. 138).
- Ephemeral storage for nonpersistent storage. For more information, see Fargate Task Storage (p. 125).

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. 125)
- Docker Volumes (p. 126)
- Bind Mounts (p. 131)
- Amazon EFS Volumes (p. 138)

Fargate Task Storage

For Fargate tasks, the following storage types are supported:

- Amazon EFS volumes for persistent storage. For more information, see Amazon EFS Volumes (p. 138).
- Ephemeral storage for nonpersistent storage.

When provisioned, each Amazon ECS task on Fargate receives the following ephemeral storage. The ephemeral storage configuration depends on which platform version the task is using. After a Fargate task stops, the ephemeral storage is deleted. For more information about Amazon ECS default service limits, see Amazon ECS service quotas (p. 612).

Fargate tasks using platform version 1.4.0 or later

All Amazon ECS on Fargate tasks using platform version 1.4.0 or later receive a minimum of 20 GB of ephemeral storage. The amount of storage is not adjustable.

For tasks using platform version 1.4.0 or later that are launched on May 28, 2020 or later, the ephemeral storage is encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key.

Fargate tasks using platform version 1.3.0 or earlier

For Amazon ECS on Fargate tasks using platform version 1.3.0 or earlier, each task receives the following ephemeral storage.

- 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.

Example task definition

To provide nonpersistent empty storage for containers in a Fargate task

In this example, you have two application containers that need to access the same scratch file storage location.

1. In the task definition `volumes` section, define a volume with the name `application_scratch`.

```
"volumes": [
  {
    "name": "application_scratch",
    "host": {}
  }
]
```

2. In the `containerDefinitions` section, create the application container definitions so they mount the nonpersistent storage.

```
"containerDefinitions": [
  {
    "name": "application1",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_scratch",
        "containerPath": "/var/scratch"
      }
    ]
  },
  {
    "name": "application2",
    "image": "my-repo/application",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "application_scratch",
        "containerPath": "/var/scratch"
      }
    ]
  }
]
```

Docker Volumes

When using Docker volumes, the built-in `local` driver or a third-party volume driver can be used. Docker volumes are managed by Docker and a directory is created in `/var/lib/docker/volumes` on the container instance that contains the volume data.

To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see 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

Docker Volume Considerations

The following should be considered 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.
- If a third-party driver is used, it should be installed and active on the container instance prior to the container agent starting. If the third-party driver is not active prior to the agent starting, you can restart the container agent using one of the following commands:
  - For the Amazon ECS-optimized Amazon Linux 2 AMI:

    ```
    sudo systemctl restart ecs
    ```

  - For the Amazon ECS-optimized Amazon Linux AMI:

    ```
    sudo stop ecs && sudo start ecs
    ```

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.

```
{
    "containerDefinitions": [
        {
            "mountPoints": [
                {
                    "sourceVolume": "string",
                    "containerPath": "/path/to/mount_volume",
                    "readOnly": boolean
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "string",
            "dockerVolumeConfiguration": {
                "scope": "string",
                "autoprovision": boolean,
                "driver": "string",
                "driverOpts": {
                    "key": "value"
                },
                "labels": {
                    "key": "value"
                }
            }
        }
    ]
}
```

`name`

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. This name is referenced in the `sourceVolume` parameter of 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 destroyed when the task is cleaned up. Docker volumes that are scoped as `shared` persist after the task stops.

`autoprovision`

Type: Boolean

Default value: `false`

Required: No

If this value is `true`, the Docker volume is created if it does not already exist. This field is only used if the `scope` is `shared`. If the `scope` is `task` then this parameter must either be omitted or set to `false`.

`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 Array

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 and use the built-in `local` driver.

```
"volumes": [
```

```
    {
        "name": "scratch",
        "dockerVolumeConfiguration" : {
            "scope": "task",
            "driver": "local",
            "labels": {
                "scratch": "space"
            }
        }
    }
]
```

2. In the `containerDefinitions` section, define a container with `mountPoints` values that reference the name of the defined volume and the `containerPath` value to mount the volume at on the container.

```
"containerDefinitions": [
    {
        "name": "container-1",
        "mountPoints": [
            {
              "sourceVolume": "scratch",
              "containerPath": "/var/scratch"
            }
        ]
    }
]
```

To provide persistent storage for 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 for tasks using the EC2 launch type.

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.

Note
For information on the task storage for Amazon ECS on Fargate tasks, see Fargate Task Storage (p. 125).

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:

```
{
   "family": "",
   ...
   "containerDefinitions" : [
      {
         "mountPoints" : [
            {
               "containerPath" : "/path/to/mount_volume",
               "sourceVolume" : "string"
            }
         ],
         "name" : "string"
      }
```

```
    ],
    ...
    "volumes" : [
       {
          "host" : {
             "sourcePath" : "string"
          },
          "name" : "string"
       }
    ]
}
```

`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`.

`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`.

`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 Array

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

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. 270). If you need this data to persist, specify a `sourcePath` value for the bind mount.

```
"volumes": [
  {
    "name": "database_scratch",
    "host": {}
  }
]
```

2. In the `containerDefinitions` section, create the database container definitions so that they mount the nonpersistent storage.

```
"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.

```
"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.

```
"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`.

```
"volumes": [
  {
    "name": "webroot",
    "host": {
      "sourcePath": "/data/webroot"
    }
  }
]
```

2. In the `containerDefinitions` section, define a container for each web server with `mountPoints` values that associate the `webroot` volume with the `containerPath` value pointing to the document root for that container.

```
"containerDefinitions": [
  {
    "name": "web-server-1",
    "image": "my-repo/ubuntu-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
        "containerPath": "/var/www/html",
        "readOnly": true
      }
    ]
  },
  {
    "name": "web-server-2",
    "image": "my-repo/sles11-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
```

```
          "containerPort": 8080,
          "hostPort": 8080
        }
      ],
      "essential": true,
      "mountPoints": [
        {
          "sourceVolume": "webroot",
          "containerPath": "/srv/www/htdocs",
          "readOnly": true
        }
      ]
    }
  ]
```

To mount volumes from another container using `volumesFrom`

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": [
        {
          "sourceVolume": "empty",
          "containerPath": "/usr/local/apache2/htdocs/empty_volume"
        },
        {
          "sourceVolume": "host_etc",
          "containerPath": "/usr/local/apache2/htdocs/host_etc"
        }
      ],
      "essential": true
    },
```

Create another container that uses `volumesFrom` to mount all of the volumes that are associated with the `web` container. All of the volumes on the `web` container are likewise mounted on the `busybox` container (including the volume specified in the Dockerfile that was used to build the `my-repo/httpd_dockerfile_volume` image).

```
    {
      "name": "busybox",
      "image": "busybox",
      "volumesFrom": [
        {
          "sourceContainer": "web"
        }
      ],
      "cpu": 100,
      "memory": 500,
      "entryPoint": [
        "sh",
        "-c"
      ],
      "command": [
        "echo $(date) > /usr/local/apache2/htdocs/empty_volume/date && echo $(date)
 > /usr/local/apache2/htdocs/host_etc/date && echo $(date) > /usr/local/apache2/htdocs/
dockerfile_volume/date"
      ],
      "essential": false
    }
  ]
}
```

When this task is run, the two containers mount the volumes, and the `command` in the `busybox` container writes the date and time to a file 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.

Amazon EFS Volumes

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.

For a tutorial, see Tutorial: Using Amazon EFS File Systems with Amazon ECS (p. 684).

Amazon EFS Volume Considerations

The following should be considered when using Amazon EFS volumes:

- For tasks using the EC2 launch type, Amazon EFS file system support was added as a public preview with Amazon ECS-optimized AMI version `20191212` with container agent version 1.35.0. However, Amazon EFS file system support entered general availability with Amazon ECS-optimized AMI version `20200319` with container agent version 1.38.0, which contained the Amazon EFS access point and IAM authorization features. It is recommended you use Amazon ECS-optimized AMI version `20200319` or later to take advantage of these features. For more information, see Amazon ECS-optimized AMI Versions (p. 198).

  Note
  If you create your own AMI, you must use container agent 1.38.0 or later, `ecs-init` version 1.38.0-1 or later, and run the following commands on your Amazon EC2 instance to enable the Amazon ECS volume plugin. The commands will be dependent on whether you are using Amazon Linux 2 or Amazon Linux as your base image.
  Amazon Linux 2

  ```
  yum install amazon-efs-utils
  systemctl enable --now amazon-ecs-volume-plugin
  ```

  Amazon Linux

  ```
  yum install amazon-efs-utils
  sudo shutdown -r now
  ```

- For tasks using the Fargate launch type, Amazon EFS file system support was added when using platform version 1.4.0 or later. For more information, see AWS Fargate Platform Versions (p. 33).
- When specifying Amazon EFS volumes in tasks using the Fargate launch type, Fargate creates a supervisor container that is responsible for managing the Amazon EFS volume. The supervisor container uses a small amount of the task's memory. The supervisor container is visible when querying the task metadata version 4 endpoint, but is not visible in CloudWatch Container Insights. For more information, see Task Metadata Endpoint version 4 (p. 292).

Using Amazon EFS Access Points

Amazon EFS access points are application-specific entry points into an EFS file system that make it easier to manage application access to shared datasets. For more information on Amazon EFS access points and how to control access to them, see Working with Amazon EFS Access Points in the *Amazon Elastic File System User Guide*.

Access points can enforce a user identity, including the user's POSIX groups, for all file system requests that are made through the access point. Access points can also enforce a different root directory for the file system so that clients can only access data in the specified directory or its subdirectories.

You can use an Amazon ECS task IAM role to enforce that specific applications use a specific access point. By combining IAM policies with access points, you can easily provide secure access to specific datasets for your applications. For more information on using task IAM roles, see IAM Roles for Tasks (p. 489).

Specifying an Amazon EFS File System in your Task Definition

In order to use Amazon EFS file system volumes for your containers, you must specify the volume and mount point configurations in your task definition. The following task definition JSON snippet shows the syntax for the `volumes` and `mountPoints` objects for a container:

```
{
    "containerDefinitions": [
        {
            "name": "container-using-efs",
            "image": "amazonlinux:2",
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "ls -la /mount/efs"
            ],
            "mountPoints": [
                {
                    "sourceVolume": "myEfsVolume",
                    "containerPath": "/mount/efs",
                    "readOnly": true
                }
            ]
        }
    ],
    "volumes": [
        {
            "name": "myEfsVolume",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1234",
                "rootDirectory": "/path/to/my/data",
                "transitEncryption": "ENABLED",
                "transitEncryptionPort": integer,
                "authorizationConfig": {
                    "accessPointId": "fsap-1234",
                    "iam": "ENABLED"
                }
            }
        }
    ]
}
```

`efsVolumeConfiguration`

Type: Object

Required: No

This parameter is specified when using Amazon EFS volumes.

`fileSystemId`

Type: String

Required: Yes

The Amazon EFS file system ID to use.

`rootDirectory`

Type: String

Required: No

The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume will be used. Specifying `/` will have the same effect as omitting this parameter.

`transitEncryption`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether or not to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. Transit encryption must be enabled if Amazon EFS IAM authorization is used. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see Encrypting Data in Transit in the *Amazon Elastic File System User Guide*.

`transitEncryptionPort`

Type: Integer

Required: No

The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you do not specify a transit encryption port, it will use the port selection strategy that the Amazon EFS mount helper uses. For more information, see EFS Mount Helper in the *Amazon Elastic File System User Guide*.

`authorizationConfig`

Type: Object

Required: No

The authorization configuration details for the Amazon EFS file system.

`accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value will be relative to the directory set for the access point. If specified, transit encryption must be enabled in the `EFSVolumeConfiguration`. For more information, see Working with Amazon EFS Access Points in the *Amazon Elastic File System User Guide*.

`iam`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether or not to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the `EFSVolumeConfiguration`. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see IAM Roles for Tasks.

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 (ENI) and a primary private IP address. 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.

Note
For information about the other available network modes for tasks, see Network mode (p. 85).

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 your tasks. Because each task gets its own ENI, 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 ENI associated with it at a given time.

For tasks that use the Fargate launch type, the task ENI that is created is fully managed by AWS Fargate. Depending on which Fargate platform version your task uses, the network traffic for your task may also use a separate Fargate-owned ENI. For more information, see Fargate Task Networking in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

For tasks that use the EC2 launch type, the task ENI that is created is fully managed by Amazon ECS. Amazon ECS creates the ENI and attaches it to the container instance with the specified security group. The task sends and receives network traffic on the ENI in the same way that Amazon EC2 instances do with their primary network interfaces. These ENIs 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 ENI that is associated with a running task. You can view the ENI attachment information for tasks in the Amazon ECS console or with the DescribeTasks API operation. When the task stops or if the service is scaled down, the task ENI is detached and deleted.

If your account, IAM user, or role has opted in to the `awsvpcTrunking` account setting and you have launched a container instance with the increased ENI density, Amazon ECS also creates and attaches a "trunk" network interface for your container instance. The trunk network is fully managed by Amazon ECS. The trunk ENI is deleted when you either terminate or deregister your container instance from the Amazon ECS cluster. For more information on opting in to the `awsvpcTrunking` account setting, see Working With Container Instances With Increased ENI Limits (p. 231).

Task Networking Considerations

There are several things to consider when using task networking.

- 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 Service-Linked Role for Amazon ECS (p. 473). 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
```

- Amazon ECS populates the hostname of a task using task networking with an Amazon-provided (internal) DNS hostname when both the `enableDnsHostnames` and `enableDnsSupport` options are enabled on your VPC. If these options are not enabled, the DNS hostname of the task will be a random hostname. For more information on the DNS settings for a VPC, see Using DNS with Your VPC in the *Amazon VPC User Guide*.
- Amazon ECS allows the launch of container instances using supported Amazon EC2 instance types with increased ENI density. When you use these instance types and opt in to the `awsvpcTrunking` account setting, newly launched container instances have higher ENI limits. This configuration allows you to place more tasks on each container instance. For more information on opting in to the `awsvpcTrunking` account setting, see Account Settings (p. 183). For more information on ENI trunking, see Elastic Network Interface Trunking (p. 230).
- Your Amazon ECS container instances require at least version 1.15.0 of the container agent to enable task networking. To take advantage of the increased ENI density with the trunking feature, your container instances require at least version 1.28.1 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. 264). If you are using an Amazon ECS-optimized AMI, your instance needs at least version 1.15.0-4 (or 1.28.1-2 for the ENI trunking feature) of the `ecs-init` package. For more information, see Amazon ECS-optimized AMIs (p. 190).
- Currently, only Linux variants of the Amazon ECS-optimized AMI, or other Amazon Linux variants with the `ecs-init` package, support task networking.
- The `awsvpc` network mode does not provide task ENIs with public IP addresses for tasks that use the EC2 launch type. To access the internet, tasks that use the EC2 launch type should 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 routed through a load balancer from within the VPC. Tasks launched within public subnets do not have access to the internet.

  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.
- Each Amazon ECS task that uses the `awsvpc` network mode receives its own ENI, which is attached to the container instance that hosts it. There is a default limit to the number of network interfaces that can be attached to an Amazon EC2 instance, and the primary network interface counts as one. For example, by default a `c5.large` instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs to the instance. Because each task using the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type. For more information on the default ENI limits for each instance type, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.
- Amazon ECS supports launching container instances with increased ENI density using supported Amazon EC2 instance types. When you use these instance types and opt in to the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks using the `awsvpc` network mode on each container instance. Using this feature, a `c5.large` instance with `awsvpcTrunking` enabled has an increased ENI limit of twelve. The container instance will have the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the container instance. So this configuration allows you to launch ten tasks on the container instance instead of the current two tasks. For more information, see Elastic Network Interface Trunking (p. 230).
- 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 ENIs that it attaches to your container instances for you. If you have attached ENIs to your container instances manually, then Amazon ECS could try to place a task on an instance with too few 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 ENIs 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 ENIs 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 ENI that is associated with a running task. To release the ENIs 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 for each 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 ENI, and they can communicate with each other over the `localhost` interface.
- Services with tasks that use the `awsvpc` network mode, such as 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 ENI, not with an Amazon EC2 instance. For more information, see Service Load Balancing (p. 376).
- If a VPC is updated, for example to change the DHCP options set it uses, and you want tasks using the VPC to pick up the changes, those tasks must be stopped and new tasks started.

Enabling Task Networking

In order for tasks to use task networking you must specify the `awsvpc` network mode in your task definition. For more information, see Network mode (p. 85). Then, when you run a task or create a service, specify a network configuration that includes one or more subnets in which to place your tasks and one or more security groups to attach to its associated ENI. 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 ENI that is provisioned for the task.

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 the containers in your task 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 Logs. 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. 237). For more information about CloudWatch Logs, see Monitoring Log Files in the *Amazon CloudWatch User Guide*.

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. For more information, see Specifying a Log Configuration in your Task Definition (p. 146).

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. 264).

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. 250).

```
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 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. 487).

Creating a Log Group

The `awslogs` log driver can send log streams to an existing log group in CloudWatch Logs or it can create a new log group on your behalf. The AWS Management Console provides an auto-configure option which creates a log group on your behalf using the task definition family name with `ecs` as the prefix. Alternatively, you can manually specify your log configuration options and specify the `awslogs-create-group` option with a value of `true` which will create the log groups on your behalf.

Note
To use the `awslogs-create-group` option to have your log group created, your IAM policy must include the `logs:CreateLogGroup` permission.

Using the Auto-configuration Feature to Create a Log Group

When registering a task definition in the Amazon ECS console, you have the option to allow Amazon ECS to auto-configure your CloudWatch logs. This option creates a log group on your behalf using the task definition family name with `ecs` as the prefix.

To use log group auto-configuration option in the Amazon ECS console

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the left navigation pane, choose **Task Definitions**, **Create new Task Definition**.
3. Select your compatibility option and choose **Next Step**.

4. Choose **Add container**.
5. In the **Storage and Logging** section, for **Log configuration**, choose **Auto-configure CloudWatch Logs**.
6. Enter your awslogs log driver options. For more information, see Specifying a Log Configuration in your Task Definition (p. 146).
7. Complete the rest of the task definition wizard.

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.

`awslogs-create-group`

Required: 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-region`

Required: 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-endpoint`

Required: No

By default, Docker uses either the `awslogs-region` log option or the detected Region to construct the remote CloudWatch Logs API endpoint. Use the `awslogs-endpoint` log option to override the default endpoint with the provided endpoint.

`awslogs-group`

Required: Yes

You must specify a log group to which the `awslogs` log driver sends its log streams. For more information, see Creating a Log Group (p. 144).

`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.

`awslogs-datetime-format`

Required: No

This option defines a multiline start pattern in Python `strftime` format. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. 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.

For more information, see awslogs-datetime-format.

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-multiline-pattern`

Required: 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.

For more information, see awslogs-multiline-pattern.

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.

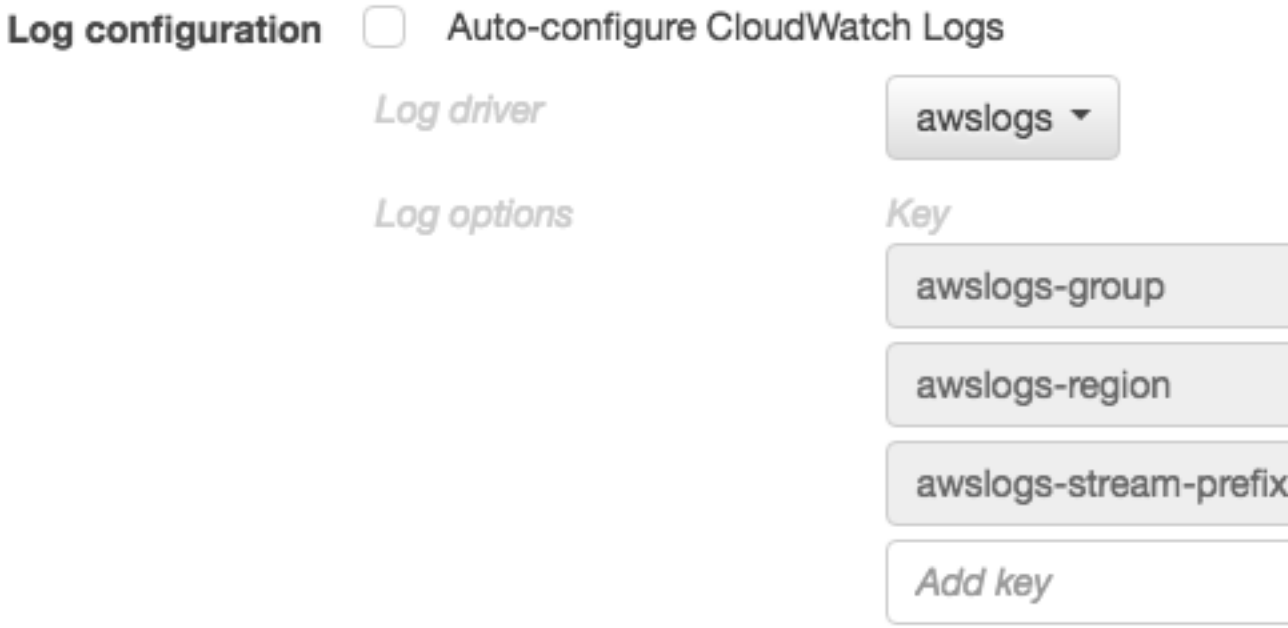
Specifying a Log Configuration in your Task Definition

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. 75).

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.

```
{
    "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",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            },
            "memory": 500,
            "cpu": 10
        },
        {
            "environment": [
                {
                    "name": "MYSQL_ROOT_PASSWORD",
                    "value": "password"
                }
            ],
            "name": "mysql",
            "image": "mysql",
            "cpu": 10,
            "memory": 500,
            "essential": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "awslogs-mysql",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "awslogs-example"
                }
            }
        }
    ],
    "family": "awslogs-example"
}
```

In the Amazon ECS console, the log configuration for the `wordpress` container is specified as shown in the image below.



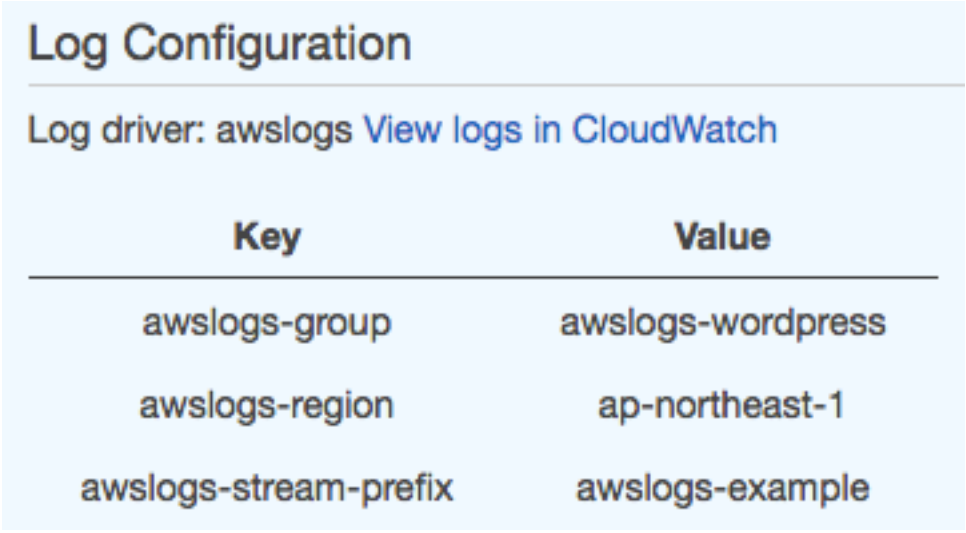
After you have registered a task definition with the `awslogs` log driver in a container definition log configuration, you can run a task or create a service with that task definition to start sending logs to CloudWatch Logs. For more information, see Running Tasks (p. 316) and Creating a service (p. 352).

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.

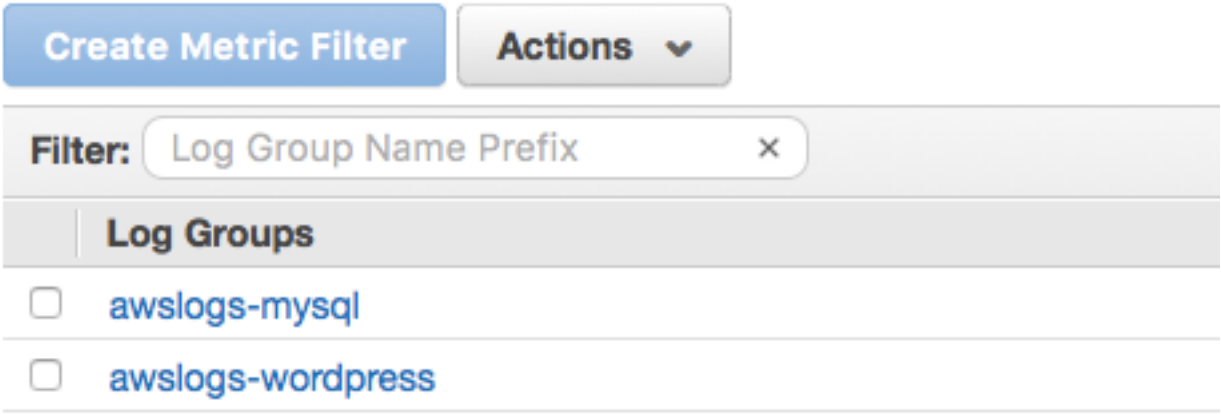
To view your CloudWatch Logs data for a container from the Amazon ECS console

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.

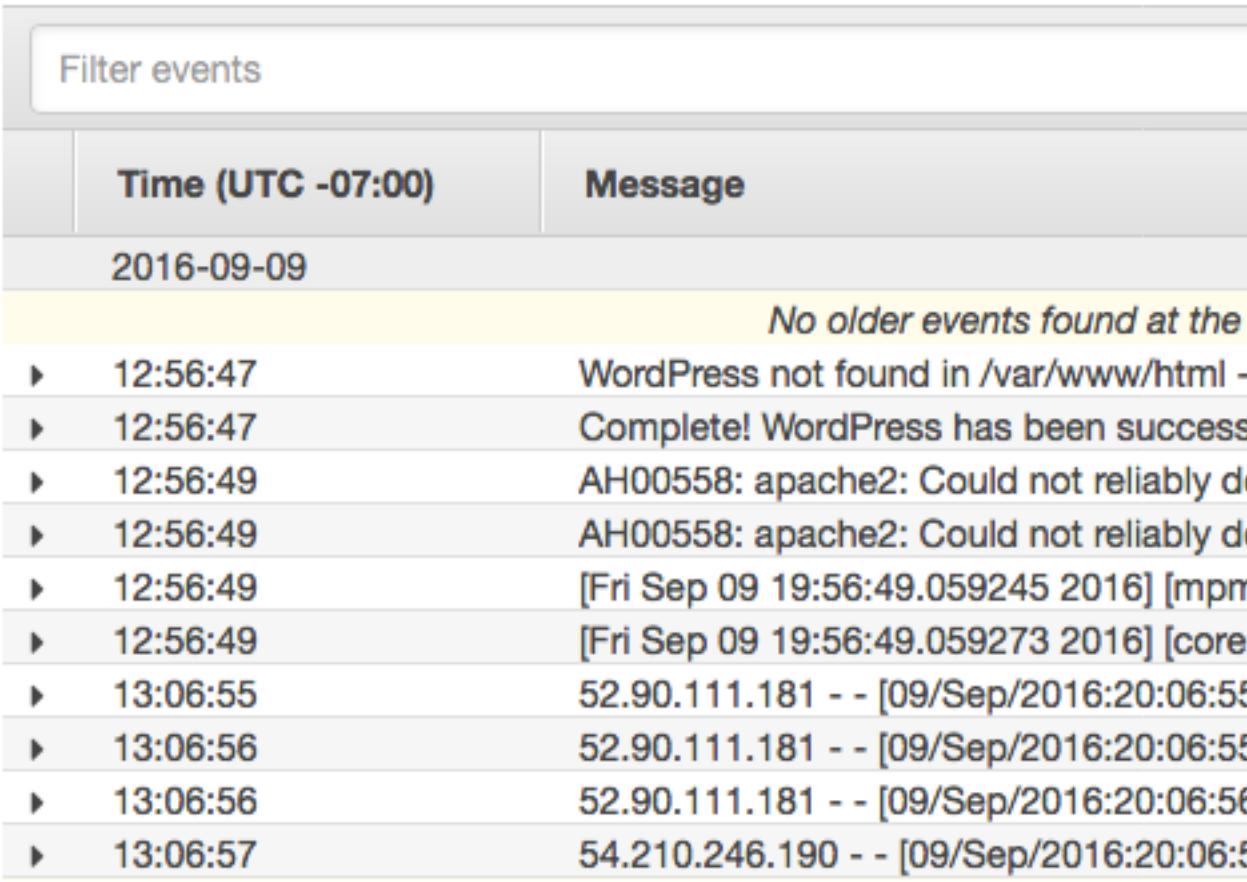


To view your CloudWatch Logs data in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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 a Log Group (p. 144).



4. Choose a log stream to view.



Custom Log Routing

FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics. FireLens works with Fluentd and Fluent Bit. We provide the AWS for Fluent Bit image or you can use your own Fluentd or Fluent Bit image.

Creating Amazon ECS task definitions with a FireLens configuration is supported using the AWS SDKs, AWS CLI, and AWS Management Console.

Considerations

The following should be considered when using FireLens for Amazon ECS:

- FireLens for Amazon ECS is supported for tasks using both the Fargate and EC2 launch types.
- FireLens for Amazon ECS is supported in AWS CloudFormation templates. For more information, see AWS::ECS::TaskDefinition FirelensConfiguration in the *AWS CloudFormation User Guide*
- For tasks that use the `bridge` network mode, the container with the FireLens configuration must start before any application containers that rely on it start. To control the start order of your containers, use dependency conditions in your task definition. For more information, see Container Dependency (p. 107).

Note
If you use dependency condition parameters in container definitions with a FireLens configuration, ensure that each container has a `START` or `HEALTHY` condition requirement.

Required IAM Permissions

To use this feature, you must create an IAM role for your tasks that provides the permissions necessary to use any AWS services that the tasks require. For example, if a container is routing logs to Kinesis Data Firehose, then the task would require permission to call the `firehose:PutRecordBatch` API. For more information, see Adding and Removing IAM Identity Permissions in the *IAM User Guide*.

The following example IAM policy adds the required permissions for routing logs to Kinesis Data Firehose.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "firehose:PutRecordBatch"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Your task may also require the Amazon ECS task execution role under the following conditions. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

- If your task uses the Fargate launch type and you are pulling container images from Amazon ECR or referencing sensitive data from AWS Secrets Manager in your log configuration, then you must include the task execution IAM role.
- If you are specifying a custom configuration file that is hosted in Amazon S3, your task execution IAM role must include the `s3:GetObject` permission for the configuration file and the `s3:GetBucketLocation` permission on the Amazon S3 bucket that the file is in. For more information, see Specifying Permissions in a Policy in the *Amazon Simple Storage Service Console User Guide*.

  The following example IAM policy adds the required permissions for retrieving a file from Amazon S3. Specify the name of your Amazon S3 bucket and configuration file name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/folder_name/config_file_name"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket"
      ]
    }
  ]
}
```

Using the AWS for Fluent Bit Image

AWS provides a Fluent Bit image with plugins for both CloudWatch Logs and Kinesis Data Firehose. We recommend using Fluent Bit as your log router because it has a lower resource utilization rate than Fluentd. For more information, see CloudWatch Logs for Fluent Bit and Amazon Kinesis Firehose for Fluent Bit.

The **AWS for Fluent Bit** image is available on Docker Hub. However, we recommend that you use the following images in Amazon ECR because they provide higher availability.

Region Name	Region	Image URI
US East (N. Virginia)	us-east-1	`906394416424.dkr.ecr.us-east-1.amazonaws.com/aws-for-fluent-bit:latest`
US East (Ohio)	us-east-2	`906394416424.dkr.ecr.us-east-2.amazonaws.com/aws-for-fluent-bit:latest`
US West (N. California)	us-west-1	`906394416424.dkr.ecr.us-west-1.amazonaws.com/aws-for-fluent-bit:latest`
US West (Oregon)	us-west-2	`906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest`
Asia Pacific (Hong Kong)	ap-east-1	`449074385750.dkr.ecr.ap-east-1.amazonaws.com/aws-for-fluent-bit:latest`
Asia Pacific (Mumbai)	ap-south-1	`906394416424.dkr.ecr.ap-south-1.amazonaws.com/aws-for-fluent-bit:latest`
Asia Pacific (Seoul)	ap-northeast-2	`906394416424.dkr.ecr.ap-northeast-2.amazonaws.com/aws-for-fluent-bit:latest`
Asia Pacific (Singapore)	ap-southeast-1	`906394416424.dkr.ecr.ap-southeast-1.amazonaws.com/`

Region Name	Region	Image URI
		aws-for-fluent-bit:latest
Asia Pacific (Sydney)	ap-southeast-2	906394416424.dkr.ecr.ap-southeast-2.amazonaws.com/aws-for-fluent-bit:latest
Asia Pacific (Tokyo)	ap-northeast-1	906394416424.dkr.ecr.ap-northeast-1.amazonaws.com/aws-for-fluent-bit:latest
Canada (Central)	ca-central-1	906394416424.dkr.ecr.ca-central-1.amazonaws.com/aws-for-fluent-bit:latest
Europe (Frankfurt)	eu-central-1	906394416424.dkr.ecr.eu-central-1.amazonaws.com/aws-for-fluent-bit:latest
Europe (Ireland)	eu-west-1	906394416424.dkr.ecr.eu-west-1.amazonaws.com/aws-for-fluent-bit:latest
Europe (London)	eu-west-2	906394416424.dkr.ecr.eu-west-2.amazonaws.com/aws-for-fluent-bit:latest
Europe (Paris)	eu-west-3	906394416424.dkr.ecr.eu-west-3.amazonaws.com/aws-for-fluent-bit:latest
Europe (Stockholm)	eu-north-1	906394416424.dkr.ecr.eu-north-1.amazonaws.com/aws-for-fluent-bit:latest
Middle East (Bahrain)	me-south-1	741863432321.dkr.ecr.me-south-1.amazonaws.com/aws-for-fluent-bit:latest
South America (São Paulo)	sa-east-1	906394416424.dkr.ecr.sa-east-1.amazonaws.com/aws-for-fluent-bit:latest

Region Name	Region	Image URI
AWS GovCloud (US-East)	us-gov-east-1	`161423150738.dkr.ecr.us-gov-east-1.amazonaws.com/aws-for-fluent-bit:latest`
AWS GovCloud (US-West)	us-gov-west-1	`161423150738.dkr.ecr.us-gov-west-1.amazonaws.com/aws-for-fluent-bit:latest`
China (Beijing)	cn-north-1	`128054284489.dkr.ecr.cn-north-1.amazonaws.com.cn/aws-for-fluent-bit:latest`
China (Ningxia)	cn-northwest-1	`128054284489.dkr.ecr.cn-northwest-1.amazonaws.com.cn/aws-for-fluent-bit:latest`

Creating a Task Definition that Uses a FireLens Configuration

To use custom log routing with FireLens you must specify the following in your task definition:

- A log router container containing a FireLens configuration. This container must be marked as essential.
- One or more application containers that contain a log configuration specifying the `awsfirelens` log driver.
- A task IAM role ARN containing the permissions needed for the task to route the logs.

When creating a new task definition using the AWS Management Console, there is a FireLens integration section that makes it easy to add a log router container. For more information, see Creating a Task Definition (p. 75).

Amazon ECS converts the log configuration and generates the Fluentd or Fluent Bit output configuration. The output configuration is mounted in the log routing container at `/fluent-bit/etc/fluent-bit.conf` for Fluent Bit and `/fluentd/etc/fluent.conf` for Fluentd.

To demonstrate how this works, the following is an example task definition example containing a log router container that uses Fluent Bit to route its logs to CloudWatch Logs and an application container that uses a log configuration to route logs to Amazon Kinesis Data Firehose.

```
{
 "family": "firelens-example-firehose",
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
```

```
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
   },
   "memoryReservation": 50
   },
   {
    "essential": true,
    "image": "httpd",
    "name": "app",
    "logConfiguration": {
     "logDriver":"awsfirelens",
     "options": {
     "Name": "firehose",
     "region": "us-west-2",
     "delivery_stream": "my-stream"
    }
   },
   "memoryReservation": 100
            }
 ]
}
```

The key-value pairs specified as options in the `logConfiguration` object are used to generate the Fluentd or Fluent Bit output configuration. The following is a code example from a Fluent Bit output definition.

```
[OUTPUT]
    Name   firehose
    Match  app-firelens*
    region us-west-2
    delivery_stream my-stream
```

Note
FireLens manages the `match` configuration. This configuration is not specified in your task definition.

Using ECS Metadata

When specifying a FireLens configuration in a task definition, you can optionally toggle the value for `enable-ecs-log-metadata`. By default, Amazon ECS adds additional fields in your log entries that help identify the source of the logs. You can disable this action by setting `enable-ecs-log-metadata` to `false`.

- `ecs_cluster` – The name of the cluster that the task is part of.
- `ecs_task_arn` – The full ARN of the task that the container is part of.
- `ecs_task_definition` – The task definition name and revision that the task is using.
- `ec2_instance_id` – The Amazon EC2 instance ID that the container is hosted on. This field is only valid for tasks using the EC2 launch type.

The following shows the syntax required when specifying an Amazon ECS log metadata setting value:

```
{
   "containerDefinitions":[
      {
         "essential":true,
         "image":"906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
         "name":"log_router",
         "firelensConfiguration":{
            "type":"fluentbit",
            "options":{
               "enable-ecs-log-metadata":"true | false"
            }
         }
      }
   ]
}
```

Specifying a Custom Configuration File

In addition to the auto-generated configuration file that FireLens creates on your behalf, you can also specify a custom configuration file. The configuration file format is the native format for the log router you're using. For more information, see Fluentd Config File Syntax and Fluent Bit Configuration Schema.

In your custom configuration file, for tasks using the `bridge` or `awsvpc` network mode, you should not set a Fluentd or Fluent Bit forward input over TCP because FireLens will add it to the input configuration.

Your FireLens configuration must contain the following options to specify a custom configuration file:

`config-file-type`

The source location of the custom configuration file. The available options are `s3` or `file`.

`config-file-value`

The source for the custom configuration file. If the `s3` config file type is used, the config file value is the full ARN of the Amazon S3 bucket and file. If the `file` config file type is used, the config file value is the full path of the configuration file that exists either in the container image or on a volume that is mounted in the container.

Important
When using a custom configuration file, you must specify a different path than the one FireLens uses. Amazon ECS reserves the `/fluent-bit/etc/fluent-bit.conf` filepath for Fluent Bit and `/fluentd/etc/fluent.conf` for Fluentd.

The following example shows the syntax required when specifying a custom configuration.

Important
To specify a custom configuration file that is hosted in Amazon S3, ensure you have created a task execution IAM role with the proper permissions. For more information, see Required IAM Permissions (p. 151).

The following shows the syntax required when specifying a custom configuration:

```
{
   "containerDefinitions":[
      {
         "essential":true,
         "image":"906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
         "name":"log_router",
         "firelensConfiguration":{
```

```
            "type":"fluentbit",
            "options":{
               "config-file-type":"s3 | file",
               "config-file-value":"arn:aws:s3:::mybucket/fluent.conf | filepath"
            }
         }
      }
   ]
}
```

Note
For tasks using the Fargate launch type, the only supported `config-file-type` value is `file`.

Using Fluent Logger Libraries

When the `awsfirelens` log driver is specified in a task definition, the ECS agent injects the following environment variables into the container:

`FLUENT_HOST`

The IP address assigned to the FireLens container.

`FLUENT_PORT`

The port that the Fluent Forward protocol is listening on.

The `FLUENT_HOST` and `FLUENT_PORT` environment variables enable you to log directly to the log router from code instead of going through `stdout`. For more information, see fluent-logger-golang on GitHub.

Filtering Logs Using Regular Expressions

Fluentd and Fluent Bit both support filtering of logs based on their content. FireLens provides a simple method for enabling this filtering. In the log configuration `options` in a container definition, you can specify the special keys `exclude-pattern` and `include-pattern` that take regular expressions as their values. The `exclude-pattern` key causes all logs that match its regular expression to be dropped. With `include-pattern`, only logs that match its regular expression are sent. These keys can be used together.

The following example demonstrates how to use this filter.

```
{
   "containerDefinitions":[
      {
         "logConfiguration":{
            "logDriver":"awsfirelens",
            "options":{
               "@type":"cloudwatch_logs",
               "log_group_name":"firelens-testing",
               "auto_create_stream":"true",
               "use_tag_as_stream":"true",
               "region":"us-west-2",
               "exclude-pattern":"^[a-z][aeiou].*$",
               "include-pattern":"^.*[aeiou]$"
            }
         }
      }
   ]
}
```

Example Task Definitions

The following are some example task definitions demonstrating common log routing options. For more examples, see Amazon ECS FireLens Examples on GitHub.

Topics

- Forwarding Logs to CloudWatch Logs (p. 158)
- Forwarding Logs to an Amazon Kinesis Data Firehose Delivery Stream (p. 159)
- Forwarding to an External Fluentd or Fluent Bit (p. 159)

Forwarding Logs to CloudWatch Logs

The following task definition example demonstrates how to specify a log configuration that forwards logs to a CloudWatch Logs log group. For more information, see What Is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*.

In the log configuration options, specify the log group name and the Region it exists in. To have Fluent Bit create the log group on your behalf, specify `"auto_create_group":"true"`. You can also specify a log stream prefix, which assists in filtering. For more information, see Fluent Bit Plugin for CloudWatch Logs.

```
{
 "family": "firelens-example-cloudwatch",
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
   },
   "memoryReservation": 50
   },
   {
    "essential": true,
    "image": "nginx",
    "name": "app",
    "logConfiguration": {
     "logDriver":"awsfirelens",
     "options": {
     "Name": "cloudwatch",
     "region": "us-west-2",
     "log_group_name": "firelens-testing-fluent-bit",
     "auto_create_group": "true",
     "log_stream_prefix": "from-fluent-bit"
    }
   },
   "memoryReservation": 100
  }
```

```
 ]
}
```

Forwarding Logs to an Amazon Kinesis Data Firehose Delivery Stream

The following task definition example demonstrates how to specify a log configuration that forwards logs to an Amazon Kinesis Data Firehose delivery stream. The Kinesis Data Firehose delivery stream must already exist. For more information, see Creating an Amazon Kinesis Data Firehose Delivery Stream in the *Amazon Kinesis Data Firehose Developer Guide*.

In the log configuration options, specify the delivery stream name and the Region it exists in. For more information, see Fluent Bit Plugin for Amazon Kinesis Firehose.

```
{
 "family": "firelens-example-firehose",
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
   },
   "memoryReservation": 50
   },
   {
    "essential": true,
    "image": "httpd",
    "name": "app",
    "logConfiguration": {
     "logDriver":"awsfirelens",
     "options": {
     "Name": "firehose",
     "region": "us-west-2",
     "delivery_stream": "my-stream"
    }
   },
   "memoryReservation": 100
            }
 ]
}
```

Forwarding to an External Fluentd or Fluent Bit

The following task definition example demonstrates how to specify a log configuration that forwards logs to an external Fluentd or Fluent Bit host. Specify the `host` and `port` for your environment.

```
{
 "family": "firelens-example-forward",
```

```
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
   },
   "memoryReservation": 50
   },
   {
    "essential": true,
    "image": "httpd",
    "name": "app",
    "logConfiguration": {
     "logDriver":"awsfirelens",
     "options": {
     "Name": "forward",
     "Host": "fluentdhost",
     "Port": "24224"
    }
   },
   "memoryReservation": 100
  }
 ]
}
```

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. 264).

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. 33).

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:

```
"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. 284).

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. 482).

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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:secret_name",
        "arn:aws:kms:<region>:<aws_account_id>:key/key_id"
      ]
    }
  ]
}
```

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:

```
{
  "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 Amazon ECS on AWS Fargate (p. 27).
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. 161).

   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:

   ```
   "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. 84).
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.

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.

Topics

- Specifying Sensitive Data Using Secrets Manager (p. 163)
- Specifying Sensitive Data Using Systems Manager Parameter Store (p. 170)

Specifying Sensitive Data Using Secrets Manager

Amazon ECS enables you to inject sensitive data into your containers by storing your sensitive data in AWS Secrets Manager secrets and then referencing them in your container definition. Sensitive data stored in Secrets Manager secrets can be exposed to a container as environment variables or as part of the log configuration.

When you inject a secret as an environment variable, you can specify a JSON key or version of a secret to inject. This process helps you control the sensitive data exposed to your container. For more information about secret versioning, see Key Terms and Concepts for AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

Topics

- Considerations for Specifying Sensitive Data Using Secrets Manager (p. 164)
- Required IAM Permissions for Amazon ECS Secrets (p. 164)
- Injecting Sensitive Data as an Environment Variable (p. 165)
- Injecting Sensitive Data in a Log Configuration (p. 168)
- Creating an AWS Secrets Manager Secret (p. 168)
- Creating a Task Definition that References Sensitive Data (p. 169)

Considerations for Specifying Sensitive Data Using Secrets Manager

The following should be considered when using Secrets Manager to specify sensitive data for containers.

- For tasks that use the Fargate launch type, the following should be considered:
  - It is only supported to inject the full contents of a secret as an environment variable. Specifying a specific JSON key or version is not supported at this time.
  - To inject the full content of a secret as an environment variable or in a log configuration, you must use platform version 1.3.0 or later. For information, see AWS Fargate Platform Versions (p. 33).
- For tasks that use the EC2 launch type, the following should be considered:
  - To inject a secret using a specific JSON key or version of a secret, your container instance must have version 1.37.0 or later of the container agent. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 264).

    To inject the full contents of a secret as an environment variable or to inject a secret in a log configuration, your container instance must have version 1.22.0 or later of the container agent.
- When using a task definition that references Secrets Manager secrets to retrieve sensitive data for your containers, if you are also using interface VPC endpoints, you must create the interface VPC endpoints for Secrets Manager. For more information, see Using Secrets Manager with VPC Endpoints in the *AWS Secrets Manager User Guide*.
- Sensitive data is injected into your container when the container is initially started. If the secret is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- For Windows tasks that are configured to use the `awslogs` logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. This can be done with User Data using the following syntax:

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE", $TRUE,
 "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers '["json-
file","awslogs"]'
</powershell>
```

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 Secrets Manager resources. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

> **Important**
> For tasks that use the EC2 launch type, you must use the ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see Amazon ECS Container Agent Configuration (p. 270).

To provide access to the Secrets Manager 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`–Required if you are referencing a Secrets Manager secret.

- `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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:<secret_name>",
        "arn:aws:kms:<region>:<aws_account_id>:key/<key_id>"
      ]
    }
  ]
}
```

Injecting Sensitive Data as an Environment Variable

Within your container definition, you can specify the following:

- The `secrets` object containing the name of the environment variable to set in the container
- The Amazon Resource Name (ARN) of the Secrets Manager secret
- Additional parameters that contain the sensitive data to present to the container

The following example shows the full syntax that must be specified for the Secrets Manager secret.

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret-name:json-key:version-
stage:version-id
```

Important
If you are using AWS Fargate, it is only supported to specify the full ARN of the secret in your task definition. Specifying a specific JSON key or version is not supported at this time.

The following section describes the additional parameters. These parameters are optional, but if you do not use them, you must include the colons `:` to use the default values. Examples are provided below for more context.

`json-key`

Specifies the name of the key in a key-value pair with the value that you want to set as the environment variable value. Only values in JSON format are supported. If you do not specify a JSON key, then the full contents of the secret is used.

`version-stage`

Specifies the staging label of the version of a secret that you want to use. If a version staging label is specified, you cannot specify a version ID. If no version stage is specified, the default behavior is to retrieve the secret with the `AWSCURRENT` staging label.

Staging labels are used to keep track of different versions of a secret when they are either updated or rotated. Each version of a secret has one or more staging labels and an ID. For more information, see Key Terms and Concepts for AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

`version-id`

Specifies the unique identifier of the version of a secret that you want to use. If a version ID is specified, you cannot specify a version staging label. If no version ID is specified, the default behavior is to retrieve the secret with the `AWSCURRENT` staging label.

Version IDs are used to keep track of different versions of a secret when they are either updated or rotated. Each version of a secret has an ID. For more information, see Key Terms and Concepts for AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

For a full tutorial on creating a Secrets Manager secret and injecting it into a container as an environment variable, see Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets (p. 657).

Example Container Definitions

The following examples show ways in which you can reference Secrets Manager secrets in your container definitions.

Example referencing a full secret

The following is a snippet of a task definition showing the format when referencing the full text of a Secrets Manager secret.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-AbCdEf"
    }]
  }]
}
```

Example referencing a specific key within a secret

The following shows an example output from a get-secret-value command that displays the contents of a secret along with the version staging label and version ID associated with it.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "VersionId": "871d9eca-18aa-46a9-8785-981dd39ab30c",
    "SecretString": "{\"username1\":\"password1\",\"username2\":\"password2\",
\"username3\":\"password3\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1581968848.921
}
```

Reference a specific key from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::"
```

```
    }]
  }]
}
```

Example referencing a specific secret version

The following shows an example output from a describe-secret command that displays the unencrypted contents of a secret along with the metadata for all versions of the secret.

```
{
    "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
    "Name": "appauthexample",
    "Description": "Example of a secret containing application authorization data.",
    "RotationEnabled": false,
    "LastChangedDate": 1581968848.926,
    "LastAccessedDate": 1581897600.0,
    "Tags": [],
    "VersionIdsToStages": {
        "871d9eca-18aa-46a9-8785-981dd39ab30c": [
            "AWSCURRENT"
        ],
        "9d4cb84b-ad69-40c0-a0ab-cead36b967e8": [
            "AWSPREVIOUS"
        ]
    }
}
```

Reference a specific version staging label from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf::AWSPREVIOUS:"
    }]
  }]
}
```

Reference a specific version ID from the previous output in a container definition by specifying the key name at the end of the ARN.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf::9d4cb84b-ad69-40c0-a0ab-cead36b967e8"
    }]
  }]
}
```

Example referencing a specific key and version staging label of a secret

The following shows how to reference both a specific key within a secret and a specific version staging label.

```
{
```

```
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1:AWSPREVIOUS:"
    }]
  }]
}
```

To specify a specific key and version ID, use the following syntax.

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-cead36b967e8"
    }]
  }]
}
```

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 the Secrets Manager secret containing the sensitive data to present to the container.

The following is a snippet of a task definition showing the format when referencing an Secrets Manager secret.

```
{
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "splunk",
      "options": {
        "splunk-url": "https://cloud.splunk.com:8080"
      },
      "secretOptions": [{
        "name": "splunk-token",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:secret_name-
AbCdEf"
      }]
    }]
  }]
}
```

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. To create your own AWS KMS CMK, you must have permissions to create CMKs in your account.
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 a Task Definition that References Sensitive Data

You can use the Amazon ECS console to create a task definition that references an Secrets Manager secret.

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**.

   Note
   This step only applies to Regions that currently support Amazon ECS using AWS Fargate. For more information, see Amazon ECS on AWS Fargate (p. 27).
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. 161).

   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:

   ```
   "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 sensitive data to inject 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 ARN of the Secrets Manager secret that contains the data to present to your container as an environment variable.

f. For sensitive data 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 of the Secrets Manager secret that contains the data to present to your log configuration as a log option.

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. 84).
h. Choose **Add**.

7. When your containers are added, choose **Create**.

Specifying Sensitive Data Using Systems Manager Parameter Store

Amazon ECS enables 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.

Topics

- Considerations for Specifying Sensitive Data Using Systems Manager Parameter Store (p. 170)
- Required IAM Permissions for Amazon ECS Secrets (p. 171)
- Injecting Sensitive Data as an Environment Variable (p. 172)
- Injecting Sensitive Data in a Log Configuration (p. 172)
- Creating an AWS Systems Manager Parameter Store Parameter (p. 173)
- Creating a Task Definition that References Sensitive Data (p. 173)

Considerations for Specifying Sensitive Data Using Systems Manager Parameter Store

The following should be considered when specifying sensitive data for containers using Systems Manager Parameter Store parameters.

- 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. 33).

- 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. 264).
- Sensitive data is injected into your container when the container is initially started. If the secret or Parameter Store parameter is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- For Windows tasks that are configured to use the `awslogs` logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. This can be done with User Data using the following syntax:

```
<powershell>
[Environment]::SetEnvironmentVariable("ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE", $TRUE,
 "Machine")
Initialize-ECSAgent -Cluster <cluster name> -EnableTaskIAMRole -LoggingDrivers '["json-
file","awslogs"]'
</powershell>
```

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 resources. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

> **Important**
> For tasks that use the EC2 launch type, you must use the ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see Amazon ECS Container Agent Configuration (p. 270).

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:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameters",
        "secretsmanager:GetSecretValue",
```

```
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:ssm:<region>:<aws_account_id>:parameter/<parameter_name>",
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:<secret_name>",
        "arn:aws:kms:<region>:<aws_account_id>:key/<key_id>"
      ]
    }
  ]
}
```

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 the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

The following is a snippet of a task definition showing the format when referencing an Systems Manager Parameter Store parameter. If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then the full ARN must be specified.

```
{
  "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 the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

Important
If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then the full ARN must be specified.

The following is a snippet of a task definition showing the format when referencing an Systems Manager Parameter Store parameter.

```
{
  "containerDefinitions": [{
    "logConfiguration": [{
      "logDriver": "fluentd",
      "options": {
        "tag": "fluentd demo"
      },
      "secretOptions": [{
        "name": "fluentd-address",
        "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter:parameter_name"
      }]
    }]
  }]
}
```

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

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
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`.
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 Sensitive Data

You can use the Amazon ECS console to create a task definition that references 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**.

   Note
   This step only applies to Regions that currently support Amazon ECS using AWS Fargate. For more information, see Amazon ECS on AWS Fargate (p. 27).
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. 161).

   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:

   ```
   "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 sensitive data to inject 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 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 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 the 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. 84).

   h. Choose **Add**.

7. When your containers are added, choose **Create**.

Specifying environment variables

Environment variables can be passed to your containers in the following ways:

- Individually using the `environment` container definition parameter. This maps to the `--env` option to **docker run**. This option is available to tasks using both the Fargate and EC2 launch types.
- In bulk, using the `environmentFiles` container definition parameter to list one or more files containing the environment variables. The file must be hosted in Amazon S3. This maps to the `--env-file` option to **docker run**. This option is only available to tasks using the EC2 launch type.

Specifying environment variables in a file enables you to bulk inject environment variables as opposed to specifying them individually. Within your container definition, specify the `environmentFiles` object with a list of Amazon S3 buckets containing your environment variable files. The files must use an `.env` file extension and there is a limit of ten files per task definition.

The following is a snippet of a task definition showing how to specify individual environment variables.

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environment": [
                {
                    "name": "variable",
                    "value": "value"
                }
            ],
            ...
        }
    ],
    ...
}
```

The following is a snippet of a task definition showing how to specify an environment variable file.

```
{
    "family": "",
    "containerDefinitions": [
        {
            "name": "",
            "image": "",
            ...
            "environmentFiles": [
                {
                    "value": "arn:aws:s3:::s3_bucket_name/envfile_object_name.env",
                    "type": "s3"
                }
            ],
            ...
        }
    ],
    ...
}
```

Considerations for specifying environment variable files

The following should be considered when specifying an environment variable file in a container definition.

- Your Amazon ECS container instances require version 1.39.0 or later of the container agent to use this feature. For information about checking your agent version and updating to the latest version, see Updating the Amazon ECS Container Agent (p. 264).
- The file must use the `.env` file extension and UTF-8 encoding.
- Each line in an environment file should contain an environment variable in `VARIABLE=VALUE` format. Spaces or quotation marks are included as part of the values. Lines beginning with # are treated as comments and are ignored. For more information on the environment variable file syntax, see Declare default environment variables in file.

  The following is an example showing the syntax that must be used.

```
#This is a comment and will be ignored
VARIABLE=VALUE
ENVIRONMENT=PRODUCTION
```

- If there are environment variables specified using the `environment` parameter in a container definition, they take precedence over the variables contained within an environment file.
- If multiple environment files are specified that contain the same variable, they are processed from the top down. It is recommended to use unique variable names.
- If an environment file is specified as a container override, it will be used and any other environment files specified in a container definition will be ignored.

Required IAM permissions

The Amazon ECS task execution role is required to use this feature. This allows the container agent to pull the environment variable file from Amazon S3. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

To provide access to the Amazon S3 objects that you create, manually add the following permissions as an inline policy to the task execution role. Use the `Resource` parameter to scope the permission to the Amazon S3 buckets that contain the environment variable files. For more information, see Adding and Removing IAM Policies.

- `s3:GetObject`
- `s3:GetBucketLocation`

An example inline policy adding the permissions is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/folder_name/env_file_name"
```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket"
      ]
    }
  ]
}
```

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. 84) and Creating a Task Definition (p. 75).

For additional task definition examples, see AWS Sample Task Definitions on GitHub.

Topics

- Example: Webserver (p. 177)
- Example: splunk Log Driver (p. 178)
- Example: fluentd Log Driver (p. 178)
- Example: gelf Log Driver (p. 179)
- Example: Amazon ECR Image and Task Definition IAM Role (p. 179)
- Example: Entrypoint with Command (p. 180)
- Example: Container Dependency (p. 180)

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"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family": "fargate-task-definition",
   "memory": "512",
   "networkMode": "awsvpc",
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

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. 163).

```
"containerDefinitions": [{
  "logConfiguration": {
   "logDriver": "splunk",
   "options": {
    "splunk-url": "https://cloud.splunk.com:8080",
    "tag": "tag_name",
   },
   "secretOptions": [{
    "name": "splunk-token",
    "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:splunk-token-KnrBkD"
}],
```

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. 163).

```
"containerDefinitions": [{
 "logConfiguration": {
  "logDriver": "fluentd",
  "options": {
   "tag": "fluentd demo"
  },
  "secretOptions": [{
   "name": "fluentd-address",
   "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:fluentd-address-
KnrBkD"
  }]
 },
 "entryPoint": [],
```

```
 "portMappings": [{
              "hostPort": 80,
              "protocol": "tcp",
              "containerPort": 80
              },
              {
  "hostPort": 24224,
  "protocol": "tcp",
  "containerPort": 24224
 }]
}],
```

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. 98).

```
"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. 489).

```
{
  "containerDefinitions": [
    {
      "name": "sample-app",
      "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-sample:v1",
      "memory": 200,
      "cpu": 10,
      "essential": true
    }
  ],
  "family": "example_task_3",
```

```
  "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
}
```

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.

```
{
  "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. 107).

```
{
  "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": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-
prod",
        "essential": true,
        "environment": [
          {
            "name": "APPMESH_VIRTUAL_NODE_NAME",
            "value": "mesh/meshName/virtualNode/virtualNodeName"
          },
          {
            "name": "ENVOY_LOG_LEVEL",
            "value": "info"
          }
        ],
        "healthCheck": {
          "command": [
            "CMD-SHELL",
            "echo hello"
          ],
          "interval": 5,
          "timeout": 2,
          "retries": 3
        }
      }
    ],
    "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
    "networkMode": "awsvpc"
}
```

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. 363).

Deregistering Task Definition Revisions

If you decide that you no longer need a specific task definition revision in Amazon ECS, you can deregister the task definition revision so that it no longer displays in your `ListTaskDefinition` API calls or in the console when you want to run a task or update a service.

When you deregister a task definition revision, it is immediately marked as `INACTIVE`. Existing tasks and services that reference an `INACTIVE` task definition revision continue to run without disruption, and existing services that reference an `INACTIVE` task definition revision can still scale up or down by modifying the service's desired count.

You cannot use an `INACTIVE` task definition revision to run new tasks or create new services, and you cannot update an existing service to reference an `INACTIVE` task definition revision (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 definition revisions remain discoverable in your account indefinitely; however, this behavior is subject to change in the future, so you should not rely on `INACTIVE` task definition revisions persisting beyond the lifecycle of any associated tasks and services.

Use the following procedure to deregister a task definition revision.

To deregister 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, choose the task definition family that contains one or more revisions that you want to deregister.
5. On the **Task Definition Name** page, select the 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.

Account Settings

Amazon ECS provides the following account settings, which enable you to opt in or out of specific features.

Amazon Resource Names (ARNs) and IDS

Resource names: `serviceLongArnFormat`, `taskLongArnFormat`, and `containerInstanceLongArnFormat`

Amazon ECS is introducing a new format for Amazon Resource Names (ARNs) and resource IDs for Amazon ECS services, tasks, and container instances. You must opt in to the new format for each resource type to use features such as resource tagging for that resource type. For more information, see Amazon Resource Names (ARNs) and IDs (p. 184).

AWSVPC Trunking

Resource name: `awsvpcTrunking`

Amazon ECS supports launching container instances with increased elastic network interface (ENI) density using supported Amazon EC2 instance types. When you use these instance types and opt in to the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks using the `awsvpc` network mode on each container instance. Using this feature, a `c5.large` instance with `awsvpcTrunking` enabled has an increased ENI limit of ten. The container instance will have a primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the container instance. Neither the primary network interface or the trunk network interface counts against the ENI limit, so this configuration allows you to launch ten tasks on the container instance instead of the current two tasks. For more information, see Elastic Network Interface Trunking (p. 230).

CloudWatch Container Insights

Resource name: `containerInsights`

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. The metrics include utilization for resources such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).

When the `containerInsights` account setting is opted in, all new clusters created after opting in will have Container Insights enabled unless you disable it during cluster creation. Individual clusters can either be enabled or disabled during creation or by using the UpdateClusterSettings API.

For clusters containing tasks or services using the EC2 launch type, your container instances must be running version 1.29.0 or later of the Amazon ECS agent to use Container Insights. For more information, see Amazon ECS Container Agent Versions (p. 259).

For each Region, you can opt in to or opt out of each account setting at the account level or for a specific IAM user or role. The available account settings to opt in to or out of include the new ARN and resource ID format and the awsvpc trunking feature.

The following are supported scenarios:

- An IAM user or role can opt in or opt out for their individual 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 to or opt out of any specific IAM role or user on the account. If the account setting for the root user is changed, it sets the default for all the IAM users and roles for which no individual account setting has been selected.

The opt in and opt out option must be selected for each account setting 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 receive the new ARN and resource ID format or the increased ENI limits. All existing resources are not affected. In order for Amazon ECS services and tasks to transition to the new ARN and resource ID formats, the service or task must be re-created. To transition a container instance to the new ARN and resource ID format or the increased ENI limits, 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 to the new format for tasks.

Topics

- Amazon Resource Names (ARNs) and IDs (p. 184)
- Viewing Account Settings (p. 185)
- Modifying Account Settings (p. 186)

Amazon Resource Names (ARNs) and IDs

When Amazon ECS resources are created, each resource is assigned a unique Amazon Resource Name (ARN) and resource identifier (ID). If you 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.

The ability to opt in to and opt out of the new Amazon Resource Name (ARN) and resource IDs is provided on a per-Region basis. Any new accounts created are opted out by default.

You can opt in or opt out of the new Amazon Resource Name (ARN) and resource ID format at any time. After you have opted in, any new resources that you create use the new format.

> **Note**
> A resource ID does not change after it's created. Therefore, opting in or out of the new format does not affect your existing resource IDs.

The following sections describe how ARN and resource ID 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, such as a service named `production`. In other cases, you must specify a resource using the Amazon Resource Name (ARN) format. The new ARN format for Amazon ECS tasks, services, and container instances includes the cluster name. For details about opting in to the new ARN format, see Modifying Account Settings (p. 186).

The following table shows both the current (old) format and the new format for each resource type.

Resource Type	ARN
Container instance	Old: `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:container-instance/`*`container-instance-id`*

Resource Type	ARN
	New: `arn:aws:ecs:`*region*`:`*aws_account_id*`:container-instance/`*cluster-name*`/`*container-instance-id*
Amazon ECS service	Old: `arn:aws:ecs:`*region*`:`*aws_account_id*`:service/`*service-name* New: `arn:aws:ecs:`*region*`:`*aws_account_id*`:service/`*cluster-name*`/`*service-name*
Amazon ECS task	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. New resource ID formats include shorter IDs for Amazon ECS tasks and container instances. The old resource ID format was 36 characters long. The new IDs are in a 32-character format that does not include any hyphens. For details about opting in to the new resource ID format, see Modifying Account Settings (p. 186).

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 or the increased ENI limits.

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, choose **Account Settings**.
4. On the **Amazon ECS ARN and resource ID settings**, **AWSVPC Trunking**, and **CloudWatch Container Insights** sections, you can view your opt-in status for each account setting 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)

```
aws ecs list-account-settings --effective-settings --region us-east-1
```

- Get-ECSAccountSetting (AWS Tools for Windows 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)

```
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)

```
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 the account settings.

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 your account settings.
3. From the dashboard, choose **Account Settings**.
4. On the **Amazon ECS ARN and resource ID settings**, **AWSVPC Trunking**, and **CloudWatch Container Insights** sections, you can select or deselect the check boxes for each account setting for the authenticated IAM user and role. Choose **Save** once finished.

   Important
   IAM users and IAM roles need the `ecs:PutAccountSetting` permission to perform this action.
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)

```
aws ecs put-account-setting-default --name serviceLongArnFormat --value enabled --
region us-east-2
```

  You can also use this command to modify the account settings for all tasks (`taskLongArnFormat`), container instances (`containerInstanceLongArnFormat`), and to opt in to the increased elastic network interface (ENI) limits for container instances (`awsvpcTrunking`). To do this, replace the `name` parameter with the corresponding resource type.

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name serviceLongArnFormat -Value enabled -Region us-
east-1 -Force
```

To modify the account settings for your 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)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --region us-
east-1
```

  You can also use this command to modify the account settings for all tasks (`taskLongArnFormat`), container instances (`containerInstanceLongArnFormat`), and to opt in to the increased elastic network interface (ENI) limits for container instances (`awsvpcTrunking`). To do this, replace the `name` parameter with the corresponding resource type.
- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
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)

```
aws ecs put-account-setting --name serviceLongArnFormat --value enabled --principal-arn
 arn:aws:iam::aws_account_id:user/principalName --region us-east-1
```

  You can also use this command to modify the account settings for all tasks (`taskLongArnFormat`), container instances (`containerInstanceLongArnFormat`), and to opt in to the increased elastic network interface (ENI) limits for container instances (`awsvpcTrunking`). To do this, replace the `name` parameter with the corresponding resource type.
- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -PrincipalArn
 arn:aws:iam::aws_account_id:user/principalName -Region us-east-1 -Force
```

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 using the EC2 launch type, your tasks 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. 188)
- Container Instance Lifecycle (p. 189)
- Check the Instance Role for Your Account (p. 190)
- Amazon ECS-optimized AMIs (p. 190)
- Retrieving Amazon ECS-Optimized AMI Metadata (p. 211)
- Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications (p. 214)
- Launching an Amazon ECS Container Instance (p. 219)
- Using Spot Instances (p. 222)
- Bootstrapping Container Instances with Amazon EC2 User Data (p. 223)
- Elastic Network Interface Trunking (p. 230)
- Connect to Your Container Instance (p. 236)
- Using CloudWatch Logs with Container Instances (p. 237)
- Container Instance Draining (p. 239)
- Container Instance Memory Management (p. 240)
- Managing Container Swap Space (p. 242)
- Manage Container Instances Remotely Using AWS Systems Manager (p. 243)
- Starting a Task at Container Instance Launch Time (p. 246)
- Deregister a Container Instance (p. 248)

Container Instance Concepts

- Your container instance must be running the Amazon ECS container agent. The container agent is able to register the instance 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. 250).
- 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. 486).
- Beginning with Linux Amazon ECS-optimized AMI version `20200430` and later, the Amazon EC2 Instance Metadata Service Version 2 (IMDSv2) is supported on your container instances. For Amazon ECS-optimized AMIs prior to version `20200430`, Amazon EC2 Instance Metadata Service Version 1 (IMDSv1) is supported. For more information, see Configuring the instance metadata service in the *Amazon EC2 User Guide for Linux Instances*.
- 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 access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

  For more information about interface VPC endpoints, see Amazon ECS interface VPC endpoints (AWS PrivateLink) (p. 503).

  If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see NAT Gateways in the *Amazon VPC User Guide* and HTTP Proxy Configuration (p. 311) in this guide. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters (p. 643).
- The type of Amazon EC2 instance that you choose for your container instances determines the resources available in your cluster. Amazon EC2 provides different instance types, each with different CPU, memory, storage, and networking capacity that you can use to run your tasks. For more information, see Amazon EC2 Instances.
- Because each container instance has unique state information that is stored locally on the container instance and within Amazon ECS:
  - You should not deregister an instance from one cluster and re-register it 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. 219).
  - You cannot stop a container instance and change its instance type. Instead, we recommend that you terminate the container instance and launch a new container instance with the desired instance size and the latest Amazon ECS-optimized Amazon Linux 2 AMI in your desired 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. 219) in this 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. 264).

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. 239).

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. 486) 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. 250).
- 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. 259).

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. 121).
- **Amazon ECS-optimized Amazon Linux AMI** – This AMI is based off of Amazon Linux. 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 is the same as the Amazon Linux AMI. For more information, see Amazon Linux AMI.
- **Amazon ECS-optimized Windows 2019 Full AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system. For more information, see Windows containers (p. 713).
- **Amazon ECS-optimized Windows 2019 Core AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system. For more information, see Windows containers (p. 713).
- **Amazon ECS-optimized Windows 1909 Core AMI** – Available for launching your Amazon ECS container instances on the Windows operating system. For more information, see Windows containers (p. 713).
- **Amazon ECS-optimized Windows 2016 Full AMI** – Available for launching your Amazon ECS container instances on the Windows operating system. For more information, see Windows containers (p. 713).

Although you can create your own container instance AMI that meets the basic specifications needed to run your containerized workloads on Amazon ECS, the Amazon ECS-optimized AMIs are preconfigured and tested on Amazon ECS by AWS engineers. It is the simplest way for you to get started and to get your containers running on AWS quickly.

The Amazon ECS-optimized AMI metadata, including the AMI name, Amazon ECS container agent version, and ECS runtime version which includes the Docker version, for each variant can be retrieved programmatically. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 211).

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

Amazon Linux 2

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

Region Name	Region	AMI ID
US East (Ohio)	`us-east-2`	View AMI ID
US East (N. Virginia)	`us-east-1`	View AMI ID
US West (N. California)	`us-west-1`	View AMI ID
US West (Oregon)	`us-west-2`	View AMI ID
Asia Pacific (Hong Kong)	`ap-east-1`	View AMI ID
Asia Pacific (Tokyo)	`ap-northeast-1`	View AMI ID
Asia Pacific (Seoul)	`ap-northeast-2`	View AMI ID

Region Name	Region	AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Amazon Linux 2 (arm64)

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

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID

Amazon Linux 2 (GPU)

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

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Amazon Linux AMI

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

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID

Region Name	Region	AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Windows Server 2019 Full

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

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID

Region Name	Region	AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Windows Server 2019 Core

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

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID

Region Name	Region	AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Windows Server 1909 Core

The following table lists the current Amazon ECS-optimized Windows 1909 Core AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID

Region Name	Region	AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Windows Server 2016 Full

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

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Topics

- Amazon ECS-optimized AMI Versions (p. 198)
- AMI Storage Configuration (p. 207)

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. 211).

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.

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
20200603	`1.40.0`	`19.03.6-ce`	`1.40.0-1`
20200430	`1.39.0`	`19.03.6-ce`	`1.39.0-1`
20200402	`1.39.0`	`18.09.9-ce`	`1.39.0-1`
20200319	`1.38.0`	`18.09.9-ce`	`1.38.0-1`
20200218	`1.37.0`	`18.09.9-ce`	`1.37.0-2`
20200205	`1.36.2`	`18.09.9-ce`	`1.36.2-1`
20200115	`1.36.1`	`18.09.9-ce`	`1.36.1-1`
20200108	`1.36.0`	`18.09.9-ce`	`1.36.0-1`
20191212	`1.35.0`	`18.09.9-ce`	`1.35.0-1`
20191114	`1.33.0`	`18.06.1-ce`	`1.33.0-1`
20191031	`1.32.1`	`18.06.1-ce`	`1.32.1-1`
20191014	`1.32.0`	`18.06.1-ce`	`1.32.0-1`
20190925	`1.32.0`	`18.06.1-ce`	`1.32.0-1`
20190913	`1.31.0`	`18.06.1-ce`	`1.31.0-1`
20190815	`1.30.0`	`18.06.1-ce`	`1.30.0-1`
20190709	`1.29.1`	`18.06.1-ce`	`1.29.1-1`
20190614	`1.29.0`	`18.06.1-ce`	`1.29.0-1`
20190607	`1.29.0`	`18.06.1-ce`	`1.29.0-1`
20190603	`1.28.1`	`18.06.1-ce`	`1.28.1-2`
20190510	`1.28.0`	`18.06.1-ce`	`1.28.0-1`
20190402	`1.27.0`	`18.06.1-ce`	`1.27.0-1`
20190301	`1.26.0`	`18.06.1-ce`	`1.26.0-1`

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20190215	1.25.3	18.06.1-ce	1.25.3-1
20190204	1.25.2	18.06.1-ce	1.25.2-1
20190127	1.25.1	18.06.1-ce	1.25.1-1
20190118	1.25.0	18.06.1-ce	1.25.0-1
20190107	1.24.0	18.06.1-ce	1.24.0-1
20181112	1.22.0	18.06.1-ce	1.22.0-1
20181016	1.20.3	18.06.1-ce	1.21.0-1

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.

Amazon ECS-optimized Amazon Linux 2 (arm64) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20200603	1.40.0	19.03.6-ce	1.40.0-1
20200430	1.39.0	19.03.6-ce	1.39.0-1
20200402	1.39.0	18.09.9-ce	1.39.0-1
20200319	1.38.0	18.09.9-ce	1.38.0-1
20200218	1.37.0	18.09.9-ce	1.37.0-2
20200205	1.36.2	18.09.9-ce	1.36.2-1
20200115	1.36.1	18.09.9-ce	1.36.1-1
20200108	1.36.0	18.09.9-ce	1.36.0-1
20191212	1.35.0	18.09.9-ce	1.35.0-1
20191114	1.33.0	18.06.1-ce	1.33.0-1
20191031	1.32.1	18.06.1-ce	1.32.1-1
20191014	1.32.0	18.06.1-ce	1.32.0-1
20190925	1.32.0	18.06.1-ce	1.32.0-1

Amazon ECS-optimized Amazon Linux 2 (arm64) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20190913	1.31.0	18.06.1-ce	1.31.0-1
20190815	1.30.0	18.06.1-ce	1.30.0-1
20190709	1.29.1	18.06.1-ce	1.29.1-1
20190617	1.29.0	18.06.1-ce	1.29.0-1
20190607	1.29.0	18.06.1-ce	1.29.0-1
20190603	1.28.1	18.06.1-ce	1.28.1-2
20190510	1.28.0	18.06.1-ce	1.28.0-1
20190403	1.27.0	18.06.1-ce	1.27.0-1
20190301	1.26.0	18.06.1-ce	1.26.0-1
20190215	1.25.3	18.06.1-ce	1.25.3-1
20190204	1.25.2	18.06.1-ce	1.25.2-1
20190127	1.25.1	18.06.1-ce	1.25.1-1
20190119	1.25.0	18.06.1-ce	1.25.0-1
20181120	1.22.0	18.06.1-ce	1.22.0-1

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.

Amazon ECS GPU-optimized AMI	Amazon ECS container agent version	Docker version	ecs-init version	NVIDIA driver version
20200603	1.40.0	19.03.6-ce	1.40.0-1	418.87.00
20200430	1.39.0	19.03.6-ce	1.39.0-1	418.87.00
20200402	1.39.0	18.09.9-ce	1.39.0-1	418.87.00
20200319	1.38.0	18.09.9-ce	1.38.0-1	418.87.00
20200218	1.37.0	18.09.9-ce	1.37.0-2	418.87.00
20200205	1.36.2	18.09.9-ce	1.36.2-1	418.87.00

Amazon ECS GPU-optimized AMI	Amazon ECS container agent version	Docker version	`ecs-init` version	NVIDIA driver version
20200115	`1.36.1`	`18.09.9-ce`	`1.36.1-1`	`418.87.00`
20200108	`1.36.0`	`18.09.9-ce`	`1.36.0-1`	`418.87.00`
20191212	`1.35.0`	`18.09.9-ce`	`1.35.0-1`	`418.87.00`
20191114	`1.33.0`	`18.06.1-ce`	`1.33.0-1`	`418.87.00`
20191031	`1.32.1`	`18.06.1-ce`	`1.32.1-1`	`418.87.00`
20191014	`1.32.0`	`18.06.1-ce`	`1.32.0-1`	`418.87.00`
20190925	`1.32.0`	`18.06.1-ce`	`1.32.0-1`	`418.87.00`
20190913	`1.31.0`	`18.06.1-ce`	`1.31.0-1`	`418.87.00`
20190815	`1.30.0`	`18.06.1-ce`	`1.30.0-1`	`418.40.04`
20190709	`1.29.1`	`18.06.1-ce`	`1.29.1-1`	`418.40.04`
20190614	`1.29.0`	`18.06.1-ce`	`1.29.0-1`	`418.40.04`
20190607	`1.29.0`	`18.06.1-ce`	`1.29.0-1`	`418.40.04`
20190603	`1.28.1`	`18.06.1-ce`	`1.28.1-2`	`418.40.04`
20190510	`1.28.0`	`18.06.1-ce`	`1.28.0-1`	`418.40.04`
20190402	`1.27.0`	`18.06.1-ce`	`1.27.0-1`	`418.40.04`
20190321	`1.26.0`	`18.06.1-ce`	`1.26.0-1`	`410.104`
20190301	`1.26.0`	`18.06.1-ce`	`1.26.0-1`	`396.26`
20190215	`1.25.3`	`18.06.1-ce`	`1.25.3-1`	`396.26`
20190204	`1.25.2`	`18.06.1-ce`	`1.25.2-1`	`396.26`
20190127	`1.25.1`	`18.06.1-ce`	`1.25.1-1`	`396.26`
20190118	`1.25.0`	`18.06.1-ce`	`1.25.0-1`	`396.26`

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.

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
2018.03.20200603	1.40.0	19.03.6-ce	1.40.0-1
2018.03.20200430	1.39.0	19.03.6-ce	1.39.0-1
2018.03.20200402	1.39.0	18.09.9-ce	1.39.0-1
2018.03.20200319	1.38.0	18.09.9-ce	1.38.0-1
2018.03.20200218	1.37.0	18.09.9-ce	1.37.0-2
2018.03.20200205	1.36.2	18.09.9-ce	1.36.2-1
2018.03.20200115	1.36.1	18.09.9-ce	1.36.1-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20191212	1.35.0	18.09.9-ce	1.35.0-1
2018.03.20191114	1.33.0	18.06.1-ce	1.33.0-1
2018.03.20191031	1.32.1	18.06.1-ce	1.32.1-1
2018.03.20191016	1.32.0	18.06.1-ce	1.32.0-1
2018.03.20191014	1.32.0	18.06.1-ce	1.32.0-1
2018.03.y	1.32.0	18.06.1-ce	1.32.0-1
2018.03.x	1.31.0	18.06.1-ce	1.31.0-1
2018.03.w	1.30.0	18.06.1-ce	1.30.0-1
2018.03.v	1.29.1	18.06.1-ce	1.29.1-1
2018.03.u	1.29.0	18.06.1-ce	1.29.0-1
2018.03.t	1.29.0	18.06.1-ce	1.29.0-1
2018.03.s	1.28.1	18.06.1-ce	1.28.1-2
2018.03.q	1.28.0	18.06.1-ce	1.28.0-1
2018.03.p	1.27.0	18.06.1-ce	1.27.0-1
2018.03.o	1.26.0	18.06.1-ce	1.26.0-1
2018.03.n	1.25.3	18.06.1-ce	1.25.3-1
2018.03.m	1.25.2	18.06.1-ce	1.25.2-1
2018.03.l	1.25.1	18.06.1-ce	1.25.1-1
2018.03.k	1.25.0	18.06.1-ce	1.25.0-1
2018.03.j	1.24.0	18.06.1-ce	1.24.0-1
2018.03.i	1.22.0	18.06.1-ce	1.22.0-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
2018.03.h	`1.21.0`	`18.06.1-ce`	`1.21.0-1`
2018.03.g	`1.20.3`	`18.06.1-ce`	`1.20.3-1`
2018.03.f	`1.20.2`	`18.06.1-ce`	`1.20.2-1`
2018.03.e	`1.20.1`	`18.03.1-ce`	`1.20.1-1`
2018.03.d	`1.20.0`	`18.03.1-ce`	`1.20.0-1`
2018.03.c	`1.19.1`	`18.03.1-ce`	`1.19.1-1`
2018.03.b	`1.19.0`	`18.03.1-ce`	`1.19.0-1`
2018.03.a	`1.18.0`	`17.12.1-ce`	`1.18.0-1`
2017.09.l	`1.17.3`	`17.12.1-ce`	`1.17.3-1`
2017.09.k	`1.17.2`	`17.12.0-ce`	`1.17.2-1`
2017.09.j	`1.17.2`	`17.12.0-ce`	`1.17.2-1`
2017.09.i	`1.17.1`	`17.09.1-ce`	`1.17.1-1`
2017.09.h	`1.17.0`	`17.09.1-ce`	`1.17.0-2`
2017.09.g	`1.16.2`	`17.09.1-ce`	`1.16.2-1`
2017.09.f	`1.16.1`	`17.06.2-ce`	`1.16.1-1`
2017.09.e	`1.16.1`	`17.06.2-ce`	`1.16.1-1`
2017.09.d	`1.16.0`	`17.06.2-ce`	`1.16.0-1`
2017.09.c	`1.15.2`	`17.06.2-ce`	`1.15.1-1`
2017.09.b	`1.15.1`	`17.06.2-ce`	`1.15.1-1`
2017.09.a	`1.15.0`	`17.06.2-ce`	`1.15.0-4`
2017.03.g	`1.14.5`	`17.03.2-ce`	`1.14.5-1`
2017.03.f	`1.14.4`	`17.03.2-ce`	`1.14.4-1`
2017.03.e	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2017.03.d	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2017.03.c	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2017.03.b	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2016.09.g	`1.14.1`	`1.12.6`	`1.14.1-1`
2016.09.f	`1.14.0`	`1.12.6`	`1.14.0-2`
2016.09.e	`1.14.0`	`1.12.6`	`1.14.0-1`
2016.09.d	`1.13.1`	`1.12.6`	`1.13.1-2`

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
2016.09.c	`1.13.1`	`1.11.2`	`1.13.1-1`
2016.09.b	`1.13.1`	`1.11.2`	`1.13.1-1`
2016.09.a	`1.13.0`	`1.11.2`	`1.13.0-1`
2016.03.j	`1.13.0`	`1.11.2`	`1.13.0-1`
2016.03.i	`1.12.2`	`1.11.2`	`1.12.2-1`
2016.03.h	`1.12.1`	`1.11.2`	`1.12.1-1`
2016.03.g	`1.12.0`	`1.11.2`	`1.12.0-1`
2016.03.f	`1.11.1`	`1.11.2`	`1.11.1-1`
2016.03.e	`1.11.0`	`1.11.2`	`1.11.0-1`
2016.03.d	`1.10.0`	`1.11.1`	`1.10.0-1`
2016.03.c	`1.10.0`	`1.11.1`	`1.10.0-1`
2016.03.b	`1.9.0`	`1.9.1`	`1.9.0-1`
2016.03.a	`1.8.2`	`1.9.1`	`1.8.2-1`
2015.09.g	`1.8.1`	`1.9.1`	`1.8.1-1`
2015.09.f	`1.8.0`	`1.9.1`	`1.8.0-1`
2015.09.e	`1.7.1`	`1.9.1`	`1.7.1-1`
2015.09.d	`1.7.1`	`1.9.1`	`1.7.1-1`
2015.09.c	`1.7.0`	`1.7.1`	`1.7.0-1`
2015.09.b	`1.6.0`	`1.7.1`	`1.6.0-1`
2015.09.a	`1.5.0`	`1.7.1`	`1.5.0-1`
2015.03.g	`1.4.0`	`1.7.1`	`1.4.0-2`
2015.03.f	`1.4.0`	`1.6.2`	`1.4.0-1`
2015.03.e	`1.3.1`	`1.6.2`	`1.3.1-1`
2015.03.d	`1.2.1`	`1.6.2`	`1.2.0-2`
2015.03.c	`1.2.0`	`1.6.2`	`1.2.0-1`
2015.03.b	`1.1.0`	`1.6.0`	`1.0-3`
2015.03.a	`1.0.0`	`1.5.0`	`1.0-1`

You can retrieve the current Amazon ECS-optimized Amazon Linux AMI using the AWS CLI with the following command:

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

Amazon ECS-optimized Windows 2019 Full AMI Versions

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

Amazon ECS-optimized Windows 2019 Full AMI	Amazon ECS container agent version	Docker version
2020.05.14	1.39.0	19.03.5
2020.01.15	1.35.0	19.03.5
2019.12.16	1.34.0	19.03.5
2019.11.25	1.34.0	19.03.4
2019.11.13	1.32.1	19.03.4
2019.10.09	1.32.0	19.03.2
2019.09.11	1.30.0	19.03.1
2019.08.16	1.29.1	19.03.1
2019.07.19	1.29.0	18.09.8
2019.05.10	1.27.0	18.09.4

The current Amazon ECS-optimized Windows 2019 Full AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

Amazon ECS-optimized Windows 2019 Core AMI Versions

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

Amazon ECS-optimized Windows 2019 Core AMI	Amazon ECS container agent version	Docker version
2020.05.14	1.39.0	19.03.5
2020.01.15	1.35.0	19.03.5
2019.12.16	1.34.0	19.03.5
2019.11.25	1.34.0	19.03.4
2019.11.13	1.32.1	19.03.4
2019.10.09	1.32.0	19.03.2

The current Amazon ECS-optimized Windows 2019 Full AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

Amazon ECS-optimized Windows 1909 Core AMI Versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows 1909 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows 1909 Core AMI	Amazon ECS container agent version	Docker version
2020.05.14	1.39.0	19.03.5
2020.01.15	1.35.0	19.03.5
2019.12.16	1.34.0	19.03.5
2019.11.25	1.34.0	19.03.4

The current Amazon ECS-optimized Windows 1909 Core AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-1909-English-Core-ECS_Optimized
```

Amazon ECS-optimized Windows 2016 Full AMI Versions

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

Amazon ECS-optimized Windows 2016 Full AMI	Amazon ECS container agent version	Docker version
2020.05.14	1.39.0	19.03.5
2020.01.15	1.35.0	19.03.5
2019.12.16	1.34.0	19.03.5
2019.11.25	1.34.0	19.03.4
2019.11.13	1.32.1	19.03.4
2019.10.09	1.32.0	19.03.2
2019.09.11	1.30.0	19.03.1
2019.08.16	1.29.1	19.03.1
2019.07.19	1.29.0	18.09.8
2019.03.07	1.26.0	18.03.1

The current Amazon ECS-optimized Windows 2016 Full AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

AMI Storage Configuration

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

Topics

- Amazon Linux 2 Storage Configuration (p. 207)
- Amazon ECS-optimized Amazon Linux AMI Storage Configuration (p. 207)

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 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. 270).
- 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. 236).
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:

   ```
    LV          VG     Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
   Convert
   ```

```
  docker-pool docker twi-aot--- 409.19g              0.16   0.08
```

5. Check the current available space in your volume group. The example below shows 612.75 GiB in the `VFree` column.

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

Output:

```
  VG     #PV #LV #SN Attr   VSize    VFree
  docker   1   1   0 wz--n- 1024.00g 612.75g
```

6. 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             0.04   0.12
```

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).

```
[ec2-user ~]$ docker info | grep "Data Space"
```

Output:

```
Data Space Used: 109.6 MB
Data Space Total: 1.539 TB
Data Space Available: 1.539 TB
```

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 Systems Manager Parameter Store API. For more information about the Systems Manager 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/<version>
```

- Amazon ECS-optimized Windows 2019 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

- Amazon ECS-optimized Windows 2016 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

The following parameter name format retrieves the metadata of the latest stable 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-04a4fb062c609f55b",
 "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:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/amzn2-ami-ecs-hvm-2.0.20181120-arm64-ebs
```

- Amazon ECS-optimized Amazon Linux AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux/amzn-ami-2017.09.l-amazon-ecs-optimized
```

Note
All versions of the Amazon ECS-optimized Amazon Linux 2 AMI are available for retrieval. Only Amazon ECS-optimized AMI versions `amzn-ami-2017.09.l-amazon-ecs-optimized` (Linux) and later can be retrieved. For more information, see Amazon ECS-optimized AMI Versions (p. 198).

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
```

- **For the Amazon ECS-optimized Amazon Linux 2 (arm64) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/arm64/recommended --region us-east-1
```

- **For the Amazon ECS GPU-optimized AMIs:**

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

- **For the Amazon ECS-optimized Amazon Linux AMIs:**

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

- **For the Amazon ECS-optimized Windows 2019 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows 2016 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized --region us-east-1
```

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. 198).

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux/amzn-ami-2017.09.l-amazon-ecs-optimized --region us-east-1
```

Example Retrieving the Amazon ECS-optimized Amazon Linux AMI metadata using the Systems Manager GetParametersByPath API

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

```
aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux/ --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`.

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

To retrieve the `image_id` value only, you can query the specific parameter value; for example:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux/recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

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 Systems Manager 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
```

Note
Referencing the Amazon ECS-optimized Windows AMIs in a AWS CloudFormation template is not supported.

Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications

Important
The Amazon SNS alert is only sent when there is a new Amazon ECS-optimized Amazon Linux AMI deployed. Generally when a new Amazon ECS-optimized Amazon Linux AMI is deployed, a

new AMI for each of the other Amazon ECS-optimized AMIs are deployed as well although there are not separate Amazon SNS alerts for them. For more information, see Amazon ECS-optimized AMIs (p. 190).

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.

AWS Region	Amazon SNS Topic ARN
us-east-1	arn:aws:sns:us-east-1:177427601217:ecs-optimized-amazon-ami-update
us-east-2	arn:aws:sns:us-east-2:177427601217:ecs-optimized-amazon-ami-update
us-west-1	arn:aws:sns:us-west-1:177427601217:ecs-optimized-amazon-ami-update
us-west-2	arn:aws:sns:us-west-2:177427601217:ecs-optimized-amazon-ami-update
ap-east-1	arn:aws:sns:ap-east-1:177427601217:ecs-optimized-amazon-ami-update
ap-northeast-1	arn:aws:sns:ap-northeast-1:177427601217:ecs-optimized-amazon-ami-update
ap-northeast-2	arn:aws:sns:ap-northeast-2:177427601217:ecs-optimized-amazon-ami-update
ap-southeast-1	arn:aws:sns:ap-southeast-1:177427601217:ecs-optimized-amazon-ami-update
ap-southeast-2	arn:aws:sns:ap-southeast-2:177427601217:ecs-optimized-amazon-ami-update

AWS Region	Amazon SNS Topic ARN
ap-south-1	arn:aws:sns:ap-south-1:177427601217:ecs-optimized-amazon-ami-update
ca-central-1	arn:aws:sns:ca-central-1:177427601217:ecs-optimized-amazon-ami-update
eu-west-1	arn:aws:sns:eu-west-1:177427601217:ecs-optimized-amazon-ami-update
eu-west-2	arn:aws:sns:eu-west-2:177427601217:ecs-optimized-amazon-ami-update
eu-west-3	arn:aws:sns:eu-west-3:177427601217:ecs-optimized-amazon-ami-update
eu-central-1	arn:aws:sns:eu-central-1:177427601217:ecs-optimized-amazon-ami-update
sa-east-1	arn:aws:sns:sa-east-1:177427601217:ecs-optimized-amazon-ami-update

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/v3/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**.
4. In the **Create Subscription** dialog box, for **Topic ARN**, paste the Amazon ECS-optimized Amazon Linux AMI update topic ARN: `arn:aws:sns:us-west-2:177427601217:ecs-optimized-amazon-ami-update`.
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:

```
aws sns --region us-west-2 subscribe --topic-arn arn:aws:sns:us-west-2:177427601217:ecs-optimized-amazon-ami-update --protocol email --notification-endpoint your_email@your_domain.com
```

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:


```
{
  "Type" : "Notification",
  "MessageId" : "e2534930-337d-5561-8636-1a2be5ba802e",
  "TopicArn" : "arn:aws:sns:us-west-2:917786371007:ecs-optimized-amazon-ami-update",
  "Message" : "{\"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-3b19b455\"},\"ap-south-1\":{\"Name\":\"amzn-ami-2017.09.j-
amazon-ecs-optimized\",\"ImageId\":\"ami-9e91cff1\"},\"ap-southeast-1\":{\"Name\":\"amzn-
ami-2017.09.j-amazon-ecs-optimized\",\"ImageId\":\"ami-f88ade84\"},\"ap-southeast-2\":
{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-optimized\",\"ImageId\":\"ami-a677b6c4\"},
\"ca-central-1\":{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-optimized\",\"ImageId\":\"ami-
db48cfbf\"},\"cn-north-1\":{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-optimized\",\"ImageId
\":\"ami-ca508ca7\"},\"eu-central-1\":{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-optimized
\",\"ImageId\":\"ami-3b7d1354\"},\"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-cad827b7\"},\"us-
east-2\":{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-optimized\",\"ImageId\":\"ami-ef64528a
\"},\"us-gov-west-1\":{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-optimized\",\"ImageId\":
\"ami-cc3cb7ad\"},\"us-west-1\":{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-optimized\",
\"ImageId\":\"ami-29b8b249\"},\"us-west-2\":{\"Name\":\"amzn-ami-2017.09.j-amazon-ecs-
optimized\",\"ImageId\":\"ami-baa236c2\"}}}]}",
  "Timestamp" : "2018-03-09T00:25:43.483Z",
  "SignatureVersion" : "1",
  "Signature" : "XWox8GDGLRiCgDOXlo/
fG9Lu/88P8S0FL6M6oQYOmUFzkucuhoblsdea3BjqdCHcWR7qdhMPQnLpN7y9iBrWVUqdAGJrukAI8athvAS+4AQD/
V/QjrhsEnlj+GaiW
+ozAu006X6GopOzFGnCtPMROjCMrMonjz7Hpv/8KRuMZR3pyQYm5d4wWB7xBPYhUMuLoZ1V8YFs55FMtgQV/
YLhSYuEu0BP1GMtLQauxDkscOtPP/vjhGQLFx1Q9LTadcQiRHtNIBxWL87PSI+BVvkin6AL7PhksvdQ7FAgHfXsit
+6p8GyOvKCqaeBG7HZhR1AbpyVka7JSNRO/6ssyrlj1g==",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-433026a4050d206028891664da859041.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:917786371007:ecs-optimized-amazon-
ami-update:8ad8798e-3bbf-4490-89b1-76136fca61dc"
}
```


The parsed `Message` value (with escaped quotes removed) is shown below:


```
{
  "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-3b19b455"
  },
  "ap-south-1": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-9e91cff1"
  },
  "ap-southeast-1": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-f88ade84"
  },
  "ap-southeast-2": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-a677b6c4"
  },
  "ca-central-1": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-db48cfbf"
  },
  "cn-north-1": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-ca508ca7"
  },
  "eu-central-1": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-3b7d1354"
  },
  "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-cad827b7"
  },
  "us-east-2": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-ef64528a"
  },
  "us-gov-west-1": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
    "ImageId": "ami-cc3cb7ad"
  },
  "us-west-1": {
    "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
```

```
          "ImageId": "ami-29b8b249"
        },
        "us-west-2": {
          "Name": "amzn-ami-2017.09.j-amazon-ecs-optimized",
          "ImageId": "ami-baa236c2"
        }
      }
    }
  ]
}
```

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 an Amazon ECS-optimized 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. 250).

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

   For more information on the latest Amazon ECS-optimized AMIs, see Amazon ECS-optimized AMIs (p. 190).
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 access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.
> For more information about interface VPC endpoints, see Amazon ECS interface VPC endpoints (AWS PrivateLink) (p. 503).
> If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see NAT Gateways in the *Amazon VPC User Guide* and HTTP Proxy Configuration (p. 311) in this guide. For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters (p. 643).

f. Select your container instance IAM role. This is usually named `ecsInstanceRole`.

> **Important**
> If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see Amazon ECS Container Instance IAM Role (p. 486).

g. (Optional) Configure your Amazon ECS container instance with user data, such as the agent environment variables from Amazon ECS Container Agent Configuration (p. 270). Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:

- By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing *your_cluster_name* with the name of your cluster.

```
#!/bin/bash
echo ECS_CLUSTER=your_cluster_name >> /etc/ecs/ecs.config
```

- If you have an `ecs.config` file in Amazon S3 and have enabled Amazon S3 read-only access to your container instance role, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing *your_bucket_name* with the name of your bucket to install the AWS CLI and write your configuration file at launch time.

> **Note**
> For more information about this configuration, see Storing Container Instance Configuration in Amazon S3 (p. 283).

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

- Specify tags for your container instance using the `ECS_CONTAINER_INSTANCE_TAGS` configuration parameter. This creates tags that are associated with Amazon ECS only, they cannot be listed using the Amazon EC2 API.

> **Important**
> If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the ECS_CONTAINER_INSTANCE_TAGS agent configuration parameter to add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
```

```
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- Specify tags for your container instance and then use the `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` configuration parameter to propagate them from Amazon EC2 to Amazon ECS

  The following is an example of a user data script that would propagate the tags associated with a container instance, as well as register the container instance with a cluster named `your_cluster_name`:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

For more information, see Bootstrapping Container Instances with Amazon EC2 User Data (p. 223).

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. 407).

   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**.

Using Spot Instances

A Spot Instance is an unused Amazon EC2 instance that is available for less than the On-Demand price. Because Spot Instances enable you to request unused EC2 instances at steep discounts, you can lower your Amazon EC2 costs significantly. The hourly price for a Spot Instance is called a Spot price. The Spot price of each instance type in each Availability Zone is set by Amazon EC2, and adjusted gradually based on the long-term supply of and demand for Spot Instances. For more information, see Spot Instances in the *Amazon EC2 User Guide for Linux Instances*.

You can register Spot Instances to your Amazon ECS clusters. For more information, see Launching an Amazon ECS Container Instance (p. 219).

Spot Instance Draining

Amazon EC2 terminates, stops, or hibernates your Spot Instance when the Spot price exceeds the maximum price for your request or capacity is no longer available. Amazon EC2 provides a Spot Instance interruption notice, which gives the instance a two-minute warning before it is interrupted. If Amazon ECS Spot Instance draining is enabled on the instance, ECS receives the Spot Instance interruption notice and places the instance in `DRAINING` status.

> **Important**
> Amazon ECS monitors for the Spot Instance interruption notices that have the `terminate` and `stop` instance-actions. If you specified either the `hibernate` instance interruption behavior when requesting your Spot Instances or Spot Fleet, then Amazon ECS Spot Instance draining is not supported for those instances.

When a container instance is set to `DRAINING`, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Service tasks on the draining container instance that are in the `PENDING` state are stopped immediately. If there are container instances in the cluster that are available, replacement service tasks are started on them.

Spot Instance draining is disabled by default and must be manually enabled. To enable Spot Instance draining for a new container instance, when launching the container instance add the following script into the **User data** field, replacing *`MyCluster`* with the name of the cluster to register the container instance to.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
EOF
```

For more information, see Launching an Amazon ECS Container Instance (p. 219).

To enable Spot Instance draining for an existing container instance

1. Connect to the Spot Instance over SSH.
2. Edit the `/etc/ecs/ecs.config` file and add the following:

```
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
```

3. Restart the `ecs` service.

   - For the Amazon ECS-optimized Amazon Linux 2 AMI:

     ```
     sudo systemctl restart ecs
     ```

   - For the Amazon ECS-optimized Amazon Linux AMI:

     ```
     sudo stop ecs && sudo start ecs
     ```

4. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see the section called "Amazon ECS Container Agent Introspection" (p. 309).

   ```
   curl http://localhost:51678/v1/metadata
   ```

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 when using the Amazon EC2 launch wizard. For more information, see Launching an Amazon ECS Container Instance (p. 219).

Topics

- Amazon ECS Container Agent (p. 223)
- Docker Daemon (p. 224)
- cloud-init-per Utility (p. 224)
- Specifying Multiple User Data Blocks Using a MIME Multi Part Archive (p. 225)
- Example Container Instance User Data Configuration Scripts (p. 226)

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. 270).

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

```
#!/bin/bash
```

```
echo "ECS_CLUSTER=MyCluster" >> /etc/ecs/ecs.config
```

If you have multiple variables to write to `/etc/ecs/ecs.config`, use the following `heredoc` format. This format writes everything between the lines beginning with **cat** and `EOF` to the configuration file.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
ECS_LOGLEVEL=debug
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. 224).

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 `heredoc` format to accomplish the same goal:

```
#cloud-boothook
cloud-init-per instance docker_options cat <<'EOF' >> /etc/sysconfig/docker
OPTIONS="${OPTIONS} --foo bar"
HTTP_PROXY=http://proxy.example.com:80/
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 Amazon EFS Volumes (p. 138)

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. 219).

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

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

# Install amazon-efs-utils
cloud-init-per once install_amazon-efs-utils yum install -y amazon-efs-utils

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

# Mount /efs
```

```
cloud-init-per once mount_efs echo -e 'fs-12345678:/ /efs efs defaults,_netdev 0 0' >> /
etc/fstab
mount -a

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

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

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

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. 219).

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

# 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/iptables/rules.v4

# 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 \
--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
ExecStop=/usr/bin/docker stop %i

[Install]
WantedBy=default.target
EOF

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. 219).

```
#!/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=cloud-final.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]
WantedBy=default.target
EOF

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

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. 38). 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>
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. 38).

You can use this script for your own container instances (provided that they are launched with a version of the **Windows_Server-2016-English-Full-Containers** AMI). Be sure to replace the *windows* line to specify your own cluster name (if you are not using a cluster called `windows`).

```
<powershell>
# Set up directories the agent uses
New-Item -Type directory -Path ${env:ProgramFiles}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS\data -Force
# Set up configuration
$ecsExeDir = "${env:ProgramFiles}\Amazon\ECS"
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", "windows", "Machine")
[Environment]::SetEnvironmentVariable("ECS_LOGFILE", "${env:ProgramData}\Amazon\ECS\log
\ecs-agent.log", "Machine")
[Environment]::SetEnvironmentVariable("ECS_DATADIR", "${env:ProgramData}\Amazon\ECS\data",
 "Machine")
# Download the agent
$agentVersion = "latest"
$agentZipUri = "https://s3.amazonaws.com/amazon-ecs-agent/ecs-agent-windows-
$agentVersion.zip"
$zipFile = "${env:TEMP}\ecs-agent.zip"
Invoke-RestMethod -OutFile $zipFile -Uri $agentZipUri
# Put the executables in the executable directory.
Expand-Archive -Path $zipFile -DestinationPath $ecsExeDir -Force
Set-Location ${ecsExeDir}
# Set $EnableTaskIAMRoles to $true to enable task IAM roles
# Note that enabling IAM roles will make port 80 unavailable for tasks.
[bool]$EnableTaskIAMRoles = $false
if (${EnableTaskIAMRoles}) {
  $HostSetupScript = Invoke-WebRequest https://raw.githubusercontent.com/aws/amazon-ecs-
agent/master/misc/windows-deploy/hostsetup.ps1
  Invoke-Expression $($HostSetupScript.Content)
}
# Install the agent service
New-Service -Name "AmazonECS" `
        -BinaryPathName "$ecsExeDir\amazon-ecs-agent.exe -windows-service" `
        -DisplayName "Amazon ECS" `
        -Description "Amazon ECS service runs the Amazon ECS agent" `
        -DependsOn Docker `
        -StartupType Manual
sc.exe failure AmazonECS reset=300 actions=restart/5000/restart/30000/restart/60000
sc.exe failureflag AmazonECS 1
Start-Service AmazonECS
</powershell>
```

Elastic Network Interface Trunking

Each Amazon ECS task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the container instance that hosts it. There is a default limit to the number of

network interfaces that can be attached to an Amazon EC2 instance, and the primary network interface counts as one. For example, by default a `c5.large` instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs to the instance. Because each task using the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type.

Amazon ECS supports launching container instances with increased ENI density using supported Amazon EC2 instance types. When you use these instance types and opt in to the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks using the `awsvpc` network mode on each container instance. Using this feature, a `c5.large` instance with `awsvpcTrunking` enabled has an increased ENI limit of twelve. The container instance will have the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the container instance. So this configuration allows you to launch ten tasks on the container instance instead of the current two tasks.

The trunk network interface is fully managed by Amazon ECS and is deleted when you either terminate or deregister your container instance from the cluster. For more information, see Task Networking with the `awsvpc` Network Mode (p. 141).

ENI Trunking Considerations

There are several things to consider when using the ENI trunking feature.

- Only Linux variants of the Amazon ECS-optimized AMI, or other Amazon Linux variants with version `1.28.1` or later of the container agent and version `1.28.1-2` or later of the ecs-init package, support the increased ENI limits. If you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. Windows containers are not supported at this time.
- Only new Amazon EC2 instances launched after opting in to `awsvpcTrunking` receive the increased ENI limits and the trunk network interface. Previously launched instances do not receive these features regardless of the actions taken.
- Amazon EC2 instances in shared subnets are not supported. They will fail to register to a cluster if they are used.
- Your Amazon ECS tasks must use the `awsvpc` network mode and the EC2 launch type. Tasks using the Fargate launch type always received a dedicated ENI regardless of how many are launched, so this feature is not needed.
- Your Amazon ECS tasks must be launched in the same Amazon VPC as your container instance. Your tasks will fail to start with an attribute error if they are not within the same VPC.
- When launching a new container instance, the instance transitions to a `REGISTERING` status while the trunk elastic network interface is provisioned for the instance. If the registration fails, the instance transitions to a `REGISTRATION_FAILED` status. You can describe the container instance and see the reason for failure in the `statusReason` parameter.
- Once the container instance is terminated, the instance transitions to a `DEREGISTERING` status while the trunk elastic network interface is deprovisioned. The instance then transitions to an `INACTIVE` status.
- If a container instance in a public subnet with the increased ENI limits is stopped and then restarted, the instance loses its public IP address, and the container agent loses its connection.

Working With Container Instances With Increased ENI Limits

Before you launch a container instance with the increased ENI limits, the following prerequisites must be completed.

- The service-linked role for Amazon ECS must be created. The Amazon ECS service-linked role provides Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster, or if you create or update a service in the AWS Management Console. For more information, see Service-Linked Role for Amazon ECS (p. 473). You can also create the service-linked role with the following AWS CLI command.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Your account or container instance IAM role must opt-in to the `awsvpcTrunking` account setting. This can be done in the following ways:
  - Any user can use the PutAccountSettingDefault API to opt-in all IAM users and roles on an account
  - A root user can use the PutAccountSetting API to opt-in the IAM user or container instance role that will register the instance with the cluster
  - A container instance role can opt itself in when the PutAccountSetting API is run on an instance prior to it being registered with a cluster

  For more information, see Account Settings (p. 183).

Once the prerequisites are met, you can launch a new container instance using one of the supported Amazon EC2 instance types, and the instance will have the increased ENI limits. For a list of supported instance types, see Supported Amazon EC2 Instance Types (p. 234). The container instance must have version `1.28.1` or later of the container agent and version `1.28.1-2` or later of the ecs-init package. If you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. For more information, see Launching an Amazon ECS Container Instance (p. 219).

To opt in in all IAM users or roles on your account to the increased ENI limits using the console

1. As the root user of the account, 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 opt in to the increased ENI limits.
3. From the dashboard, choose **Account Settings**.
4. For **IAM user or role**, ensure your root user or container instance IAM role is selected.
5. For **AWSVPC Trunking**, select the check box. Choose **Save** once finished.

   Important
   IAM users and IAM roles need the `ecs:PutAccountSetting` permission to perform this action.
6. On the confirmation screen, choose **Confirm** to save the selection.

To opt in all IAM users or roles on your account to the increased ENI limits using the command line

Any user on an account can 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)

```
aws ecs put-account-setting-default \
      --name awsvpcTrunking \
      --value enabled \
      --region us-east-1
```

- Write-ECSAccountSettingDefault (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name awsvpcTrunking -Value enabled -Region us-east-1 -
Force
```

To opt in an IAM user or container instance IAM role to the increased ENI limits as the root user using the command line

The root user on an account can use one of the following commands and specify the ARN of the principal IAM user or container instance IAM role in the request to modify the account settings.

- put-account-setting (AWS CLI)

  The following example is for modifying the account setting of a specific IAM user:

```
aws ecs put-account-setting \
      --name awsvpcTrunking \
      --value enabled \
      --principal-arn arn:aws:iam::aws_account_id:user/userName \
      --region us-east-1
```

  The following example is for modifying the account setting of a specific container instance IAM role:

```
aws ecs put-account-setting \
      --name awsvpcTrunking \
      --value enabled \
      --principal-arn arn:aws:iam::aws_account_id:role/ecsInstanceRole \
      --region us-east-1
```

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

  The following example is for modifying the account setting of a specific IAM user:

```
Write-ECSAccountSetting -Name awsvpcTrunking -Value enabled -PrincipalArn
 arn:aws:iam::aws_account_id:user/userName -Region us-east-1 -Force
```

  The following example is for modifying the account setting of a specific container instance IAM role:

```
Write-ECSAccountSetting -Name awsvpcTrunking -Value enabled -PrincipalArn
 arn:aws:iam::aws_account_id:role/ecsInstanceRole -Region us-east-1 -Force
```

To view your container instances with increased ENI limits with the AWS CLI

Each container instance has a default network interface, referred to as a trunk network interface. Use the following command to list your container instances with increased ENI limits by querying for the `ecs.awsvpc-trunk-id` attribute, which indicates it has a trunk network interface.

- list-attributes (AWS CLI)

```
aws ecs list-attributes \
      --target-type container-instance \
      --attribute-name ecs.awsvpc-trunk-id \
      --cluster cluster_name \
      --region us-east-1
```

- Get-ECSAttributeList (AWS Tools for Windows PowerShell)

```
Get-ECSAttributeList -TargetType container-instance -AttributeName ecs.awsvpc-trunk-id -
Region us-east-1
```

Supported Amazon EC2 Instance Types

The following shows the supported Amazon EC2 instance types and how many tasks using the `awsvpc` network mode can be launched on each instance type before and after opting in to the `awsvpcTrunking` account setting. For the elastic network interface (ENI) limits on each instance type, add one to the current task limit, as the primary network interface counts against the limit, and add two to the new task limit, as both the primary network interface and the trunk network instance count again the limit.

Important
Although other instance types are supported in the same instance family, the `c5n`, `m5n`, `m5dn`, `r5n`, and `r5dn` instance types are not supported.

Instance Type	Current Task Limit per Instance	New Task Limit per Instance
a1 instance family		
a1.medium	1	10
a1.large	2	10
a1.xlarge	3	20
a1.2xlarge	3	40
a1.4xlarge	7	60
c5 instance family		
c5.large	2	10
c5.xlarge	3	20
c5.2xlarge	3	40
c5.4xlarge	7	60
c5.9xlarge	7	60
c5.18xlarge	14	120
c5d.large	2	10
c5d.xlarge	3	20
c5d.2xlarge	3	40
c5d.4xlarge	7	60
c5d.9xlarge	7	60
c5d.18xlarge	14	120
m5 instance family		
m5.large	2	10

Instance Type	Current Task Limit per Instance	New Task Limit per Instance
m5.xlarge	3	20
m5.2xlarge	3	40
m5.4xlarge	7	60
m5.12xlarge	7	60
m5.24xlarge	14	120
m5a.large	2	10
m5a.xlarge	3	20
m5a.2xlarge	3	40
m5a.4xlarge	7	60
m5a.12xlarge	7	60
m5a.24xlarge	14	120
m5ad.large	2	10
m5ad.xlarge	3	20
m5ad.2xlarge	3	40
m5ad.4xlarge	7	60
m5ad.12xlarge	7	60
m5ad.24xlarge	14	120
m5d.large	2	10
m5d.xlarge	3	20
m5d.2xlarge	3	40
m5d.4xlarge	7	60
m5d.12xlarge	7	60
m5d.24xlarge	14	120
p3 instance family		
p3.2xlarge	3	40
p3.8xlarge	7	60
p3.16xlarge	7	120
r5 instance family		
r5.large	2	10
r5.xlarge	3	20
r5.2xlarge	3	40

Instance Type	Current Task Limit per Instance	New Task Limit per Instance
r5.4xlarge	7	60
r5.12xlarge	7	60
r5.24xlarge	14	120
r5a.large	2	10
r5a.xlarge	3	20
r5a.2xlarge	3	40
r5a.4xlarge	7	60
r5a.12xlarge	7	60
r5a.24xlarge	14	120
r5ad.large	2	10
r5ad.xlarge	3	20
r5ad.2xlarge	3	40
r5ad.4xlarge	7	60
r5ad.12xlarge	7	60
r5ad.24xlarge	14	120
r5d.large	2	10
r5d.xlarge	3	20
r5d.2xlarge	3	40
r5d.4xlarge	7	60
r5d.12xlarge	7	60
r5d.24xlarge	14	120

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. For more information, see Launching an Amazon ECS Container Instance (p. 219).

- 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 awslogs Log Driver (p. 143). For more information about CloudWatch Logs, see Monitoring Log Files in the *Amazon CloudWatch User Guide*.

Topics

- CloudWatch Logs IAM Policy (p. 238)
- Installing and Configuring the CloudWatch Agent (p. 238)
- Viewing CloudWatch Logs (p. 239)

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

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**, **JSON**.
4. Enter the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

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`

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. 486) to create the role.
4. Choose **Permissions**, **Attach policies**.
5. To narrow the available policies to attach, for **Filter**, type **ECS-CloudWatchLogs**.
6. Select the **ECS-CloudWatchLogs** policy and choose **Attach policy**.

Installing and Configuring the CloudWatch Agent

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

For more information, see Download and Configure the CloudWatch Agent Using the Command Line in the *Amazon CloudWatch User Guide*.

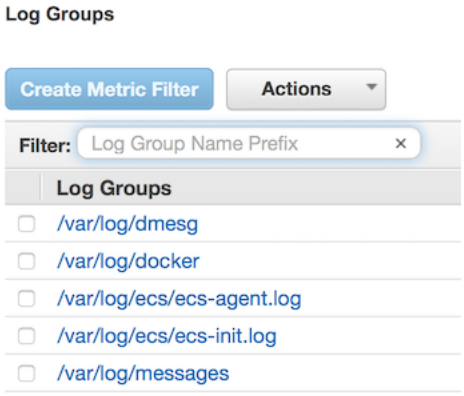
Viewing CloudWatch Logs

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.

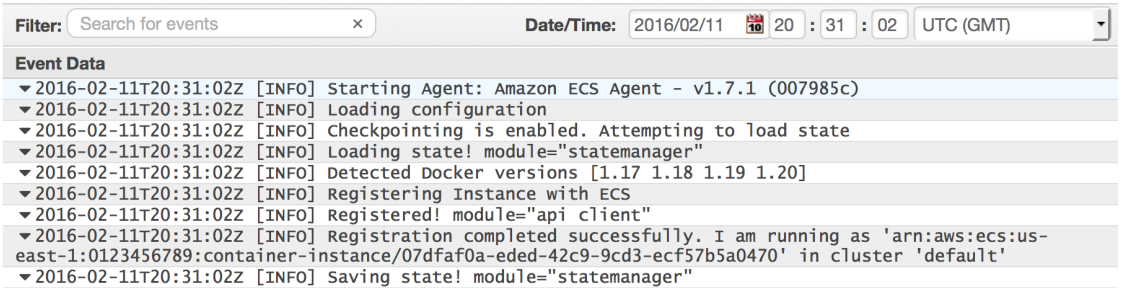
Note
New instance launches may take a few minutes to send data to CloudWatch Logs.

To view your CloudWatch Logs data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Logs**, **Log groups**.



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.



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. 340).

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. 241).

The Amazon ECS container agent uses the Docker `ReadMemInfo()` function to query the total memory available to the operating system. Both Linux and Windows provide command line utilities to determine the total memory.

Example - Determine Linux total memory

The **free** command returns the total memory that is recognized by the operating system.

```
$ free -b
```

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

```
             total       used       free     shared    buffers     cached
Mem:    8373026816  348180480 8024846336      90112   25534464  205418496
-/+ buffers/cache:  117227520 8255799296
```

This instance has 8373026816 bytes of total memory, which translates to 7985 MiB available for tasks.

Example - Determine Windows total memory

The **wmic** command returns the total memory that is recognized by the operating system.

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

Example output for an `m4.large` instance running the Amazon ECS-optimized Windows 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. 270) and Bootstrapping Container Instances with Amazon EC2 User Data (p. 223).

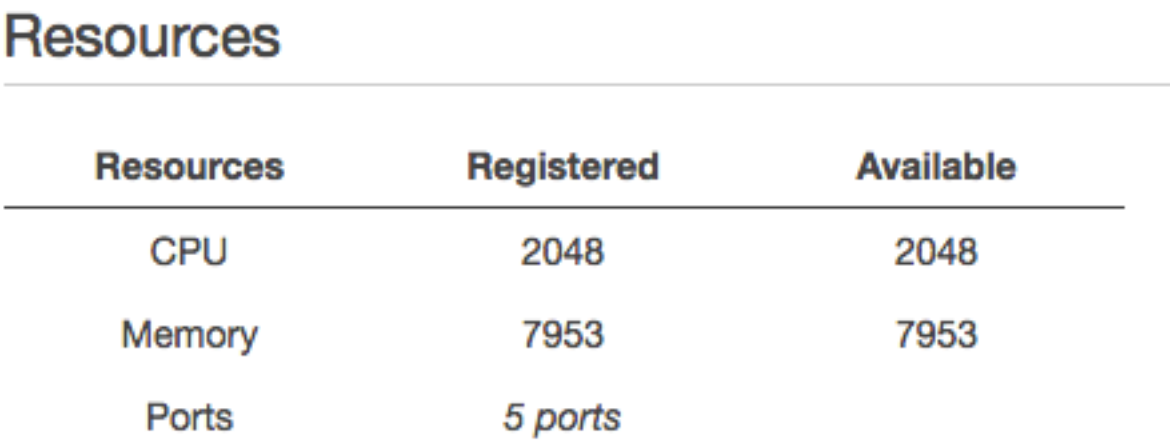
Viewing Container Instance Memory

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.



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 Swap Space

Amazon ECS enables you to control the usage of swap memory space on your Linux container instances at the container level. Using a per-container swap configuration, each container within a task definition can have swap enabled or disabled, and for those that have it enabled, the maximum amount of swap space used can be limited. For example, latency-critical containers can have swap disabled, whereas containers with high transient memory demands can have swap turned on to reduce the chances of out-of-memory errors when the container is under load.

The swap configuration for a container is managed by the following container definition parameters:

`maxSwap`

The total amount of swap memory (in MiB) a container can use. This parameter will be translated to the `--memory-swap` option to docker run where the value would be the sum of the container memory plus the `maxSwap` value.

If a `maxSwap` value of `0` is specified, the container will not use swap. Accepted values are `0` or any positive integer. If the `maxSwap` parameter is omitted, the container will use the swap configuration for the container instance it is running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

`swappiness`

This allows you to tune a container's memory swappiness behavior. A `swappiness` value of `0` will cause swapping to not happen unless absolutely necessary. A `swappiness` value of `100` will cause

pages to be swapped very aggressively. Accepted values are whole numbers between `0` and `100`. If the `swappiness` parameter is not specified, a default value of `60` is used. If a value is not specified for `maxSwap` then this parameter is ignored. This parameter maps to the `--memory-swappiness` option to docker run.

The following is an example showing the JSON syntax:

```
"containerDefinitions": [{
        ...
        "linuxParameters": {
            "maxSwap": integer,
            "swappiness": integer
        },
        ...
}]
```

Container Swap Considerations

Consider the following when you use a per-container swap configuration.

- Swap space must be enabled and allocated on the container instance for the containers to use.

  Note
  The Amazon ECS-optimized AMIs do not have swap enabled by default. You must enable swap on the instance to use this feature. For more information, see Instance Store Swap Volumes in the *Amazon EC2 User Guide for Linux Instances* or How do I allocate memory to work as swap space in an Amazon EC2 instance by using a swap file?.
- The swap space container definition parameters are only supported for task definitions using the EC2 launch type.
- This feature is only supported for Linux containers.
- If the `maxSwap` and `swappiness` container definition parameters are omitted from a task definition, each container will have a default `swappiness` value of `60` and the total swap usage will be limited to two times the memory reservation of the container.

Manage Container Instances Remotely Using AWS Systems Manager

You can use the Run Command capability in AWS Systems Manager to securely and remotely manage the configuration of your Amazon ECS container instances. Run Command provides a simple way to perform common administrative tasks without logging on locally to the instance. You can manage configuration changes across your clusters by simultaneously executing commands on multiple container instances. Run Command reports the status and results of each command.

Here are some examples of the types of tasks you can perform with Run Command:

- Install or uninstall packages.
- Perform security updates.
- Clean up Docker images.
- Stop or start services.
- View system resources.
- View log files.

- Perform file operations.

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 not comprehensive. For more information about Run Command, see AWS Systems Manager Run Command in the *AWS Systems Manager User Guide*.

Topics

- Run Command IAM Policy (p. 244)
- Installing SSM Agent on an Amazon ECS-Optimized AMI (p. 244)
- Using Run Command (p. 245)

Run Command IAM Policy

Before you can send commands to your container instances with Run Command, you must attach an IAM policy that allows `ecsInstanceRole` to have access to the Systems Manager APIs. The following procedure describes how to attach the Systems Manager managed policies to your container instance role so that instances launched with this role can use Run Command.

To attach the Systems Manager policies 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. 486) to create the role.
4. Choose the **Permissions** tab.
5. Choose **Attach policies**.
6. To narrow the available policies to attach, for **Filter**, type `SSM`.
7. In the list of policies, select the box next **AmazonSSMManagedInstanceCore**. This policy enables you to provide the minimum permissions that are necessary to use Systems Manager.

   For information about other policies you can provide for Systems Manager operations, see Create an IAM Instance Profile for Systems Manager in the *AWS Systems Manager User Guide*.
8. Choose **Attach Policy**.

Installing SSM Agent on an Amazon ECS-Optimized AMI

After you attach Systems Manager policies to your `ecsInstanceRole`, you can install AWS Systems Manager Agent (SSM Agent) on your container instances. SSM Agent is Amazon software that can be installed and configured on an Amazon EC2 instance, an on-premises server, or a virtual machine (VM). SSM Agent makes it possible for Systems Manager to update, manage, and configure these resources. SSM Agent processes Run Command requests and configures the instances that are specified in the request. Use the following procedures to install SSM Agent on your Amazon ECS-optimized AMI container instances.

> **Note**
> SSM Agent is available in all Regions that Systems Manager is available in. (For a list of supported Regions, see the **Region** column in the AWS Systems Manager Table of Regions and Endpoints topic in the *AWS General Reference*.) Each Region has its own region-specific download URL, but the following commands use your current specified AWS Region. For more

information about SSM Agent, see Working with SSM Agent in the *AWS Systems Manager User Guide*.

To manually install SSM Agent on existing Amazon ECS-optimized AMI container instances

1. Connect to your container instance. (p. 236)
2. Run the following command to install the SSM Agent RPM.

```
[ec2-user ~]$ sudo yum install -y amazon-ssm-agent
```

To install 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. 219), but in Step 6.g (p. 220), 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.

```
#!/bin/bash
# Install the SSM Agent RPM
yum install -y amazon-ssm-agent
```

Using Run Command

After you attach Systems Manager managed policies to your `ecsInstanceRole`, and install SSM Agent on your container instances, you can start using Run Command to send commands to your container instances. For information about running commands and shell scripts on your instances and viewing the resulting output, see Running Commands Using Systems Manager Run Command and Run Command Walkthroughs in the *AWS Systems Manager User Guide.*

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 Systems Manager managed policies to your `ecsInstanceRole`. (p. 244)
2. Install SSM Agent on your container instances. For more information, see Installing SSM Agent on an Amazon ECS-Optimized AMI (p. 244).
3. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager.
4. In the left navigation pane, choose **Run Command**, and then choose **Run command**.
5. For **Command document**, choose **AWS-RunShellScript**.
6. In the **Commands** section, enter the command or commands to send to your container instances. In this example, the following command updates the instance software:

```
$ yum update -y
```

7. In the **Target instances** section, select the boxes next to the container instances where you want to run the update command.
8. Choose **Run** to send the command to the specified instances.
9. (Optional) Choose the refresh icon to monitor the command status.
10. (Optional) In **Targets and output**, choose the button next to the instance ID, and then choose **View output**.

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. 75).
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. 486).
   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   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. 486) 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**, **Elastic 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. 219), but in Step 6.g (p. 220) 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.

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

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

#!/bin/bash
# Specify the cluster that the container instance should register into
cluster=your_cluster_name
```

```
# Write the cluster configuration variable to the ecs.config file
# (add any other configuration variables here also)
echo ECS_CLUSTER=$cluster >> /etc/ecs/ecs.config

START_TASK_SCRIPT_FILE="/etc/ecs/ecs-start-task.sh"
cat <<- 'EOF' > ${START_TASK_SCRIPT_FILE}
 exec 2>>/var/log/ecs/ecs-start-task.log
 set -x

 # Install prerequisite tools
 yum install -y jq aws-cli

 # Wait for the ECS service to be responsive
 until curl -s http://localhost:51678/v1/metadata
 do
  sleep 1
 done

 # Grab the container instance ARN and AWS Region from instance metadata
 instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F/ '{print $NF}' )
 cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '. | .Cluster' | awk -F/
 '{print $NF}' )
 region=$(curl -s http://localhost:51678/v1/metadata | jq -r '.
 | .ContainerInstanceArn' | awk -F: '{print $4}')

 # Specify the task definition to run at launch
 task_definition=my_task_def

 # Run the AWS CLI start-task command to start your task on this container instance
 aws ecs start-task --cluster $cluster --task-definition $task_definition --container-
instances $instance_arn --started-by $instance_arn --region $region
EOF

# Write systemd unit file
UNIT="ecs-start-task.service"
cat <<- EOF > /etc/systemd/system/${UNIT}
      [Unit]
      Description=ECS Start Task
      Requires=ecs.service
      After=ecs.service

      [Service]
      Restart=on-failure
      RestartSec=30
      ExecStart=/usr/bin/bash ${START_TASK_SCRIPT_FILE}

      [Install]
      WantedBy=default.target
EOF

# Enable our ecs.service dependent service with `--no-block` to prevent systemd
 deadlock
# See https://github.com/aws/amazon-ecs-agent/issues/1707
systemctl enable --now --no-block "${UNIT}"
----BOUNDARY----
```

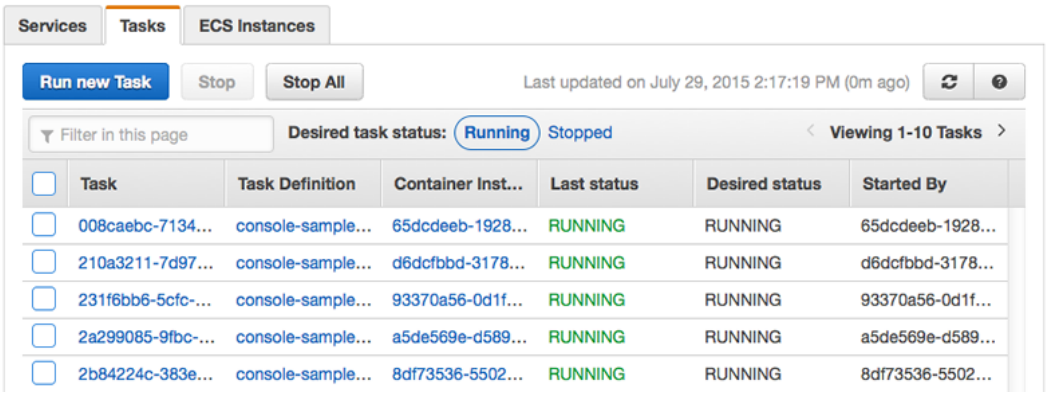
4. Verify that your container instances launch into the correct cluster and that your tasks have started.

   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 `/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. 219).

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 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 : *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
For tasks using the Fargate launch type and platform version 1.3.0 and prior, the Amazon ECS container agent is installed on the AWS managed infrastructure used for these tasks. 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. For tasks using the Fargate and platform version 1.4.0 and later, the Fargate container agent is used. For more information, see AWS Fargate platform versions in the *Amazon Elastic Container Service User Guide for AWS Fargate*

Topics

- Installing the Amazon ECS Container Agent (p. 250)
- Amazon ECS Container Agent Versions (p. 259)
- Updating the Amazon ECS Container Agent (p. 264)
- Amazon ECS Container Agent Configuration (p. 270)
- Private Registry Authentication for Container Instances (p. 284)
- Automated Task and Image Cleanup (p. 287)
- Amazon ECS Container Metadata File (p. 288)
- Amazon ECS Task Metadata Endpoint (p. 292)
- Amazon ECS Container Agent Introspection (p. 309)
- HTTP Proxy Configuration (p. 311)

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. The Amazon ECS container agent is included in the Amazon ECS-optimized AMIs and does not require installation.

- 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. 251).
- 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. 251).
- 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. 252)

Note
The systemd units for both ECS and Docker services have a directive to wait for `cloud-init` to finish before starting both services. The `cloud-init` process is not considered finished until

your Amazon EC2 user data has finished running. Therefore, starting ECS or Docker via Amazon EC2 user data may cause a deadlock. To start the container agent using Amazon EC2 user data you can use `systemctl enable --now --no-block ecs.service`.

Installing the Amazon ECS Container Agent on 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. 486).
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.

```
[ec2-user ~]$ sudo amazon-linux-extras disable docker
```

4. Install and enable the `ecs` Amazon Linux extra repository.

```
[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. 309).

```
[ec2-user ~]$ curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Note
If you get no response, ensure that you associated the Amazon ECS container instance IAM role when launching the instance. For more information, see Amazon ECS Container Instance IAM Role (p. 486).

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. 486).
2. Connect to your instance.
3. Install the `ecs-init` package. For more information about `ecs-init`, see the source code on GitHub.

```
[ec2-user ~]$ sudo yum install -y ecs-init
```

4. Start the Docker daemon.

```
[ec2-user ~]$ sudo service docker start
```

Output:

```
Starting cgconfig service:                                 [  OK  ]
Starting docker:                                  [  OK  ]
```

5. Start the `ecs-init` upstart job.

```
[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. 309).

```
[ec2-user ~]$ curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

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.

Region	Region Name	Container agent	Container agent signature
us-east-2	US East (Ohio)	ECS container agent	PGP signature
us-east-1	US East (N. Virginia)	ECS container agent	PGP signature
us-west-1	US West (N. California)	ECS container agent	PGP signature
us-west-2	US West (Oregon)	ECS container agent	PGP signature
ap-east-1	Asia Pacific (Hong Kong)	ECS container agent	PGP signature
ap-northeast-1	Asia Pacific (Tokyo)	ECS container agent	PGP signature
ap-northeast-2	Asia Pacific (Seoul)	ECS container agent	PGP signature
ap-south-1	Asia Pacific (Mumbai)	ECS container agent	PGP signature
ap-southeast-1	Asia Pacific (Singapore)	ECS container agent	PGP signature

Region	Region Name	Container agent	Container agent signature
ap-southeast-2	Asia Pacific (Sydney)	ECS container agent	PGP signature
ca-central-1	Canada (Central)	ECS container agent	PGP signature
eu-central-1	Europe (Frankfurt)	ECS container agent	PGP signature
eu-west-1	Europe (Ireland)	ECS container agent	PGP signature
eu-west-2	Europe (London)	ECS container agent	PGP signature
eu-west-3	Europe (Paris)	ECS container agent	PGP signature
sa-east-1	South America (São Paulo)	ECS container agent	PGP signature
us-gov-east-1	AWS GovCloud (US-East)	ECS container agent	PGP signature
us-gov-west-1	AWS GovCloud (US-West)	ECS container agent	PGP signature

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. 486).
2. Connect to your instance.
3. Install the latest version of Docker on your instance.

   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.

   ```
   ubuntu:~$ sudo docker version
   ```

   Output:

   ```
   Client version: 1.4.1
   Client API version: 1.16
   Go version (client): go1.3.3
   Git commit (client): 5bc2ff8
   OS/Arch (client): linux/amd64
   Server version: 1.4.1
   Server API version: 1.16
   Go version (server): go1.3.3
   Git commit (server): 5bc2ff8
   ```

   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. 489).

```
ubuntu:~$ sudo apt-get install iptables-persistent
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 -m 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
```

   For more information about these and other agent runtime options, see Amazon ECS Container Agent Configuration (p. 270).

   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. 283) and Private Registry Authentication for Tasks (p. 160).

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. 226).

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. 270).

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.

```
ubuntu:~$ curl -o ecs-agent.tar https://s3.amazonaws.com/amazon-ecs-agent-us-east-1/ecs-agent-latest.tar
```

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:

```
https://s3.amazonaws.com/amazon-ecs-agent-us-east-1/ecs-agent-v1.18.0.tar
https://s3.amazonaws.com/amazon-ecs-agent-us-east-1/ecs-agent-c0defea9.tar
```

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 \
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. 259).

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. 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

mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2fl74lmavr4Vg
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
N+6pHBJrKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO
2K7oVX8Oc3j5+ZilhL/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNIcO
lFTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn
5lBPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq
lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQq0osYXaQbvlnSWOoc1y/5Zb
zizHTJIhLtUyls9WisP2s0emeHZicVMfW61EgPrJAiupgc7kyZvFt4YwfwARAQAB
tCRBbWF6b24gRUNTIDxlY3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhwEEAECAAYF
AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPLt5bV1+rkqOlw+0gZ4Q/ue/YbWtOAt4qY0OcEo0HgcnaX
lsB827QIfZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmazA
McWB4HUMNrhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF
4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMZpVkUr+C252IaH1HZFEz+TvBVQM
Y8OWWxmIJW+J6evjo3N1eO19UHv71jvoF8zljbI4bsL2c+QTJmOv7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBBwNJzQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItenatK
lEJQhYtyVXcBlh6Yn/wzNg2NWOwb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPKmyGopsmN/reMuilPdINb249nA0dzoN+nj+tTFOYCIaLaFyjs
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX
gAAKCRC86dmkLVF4T9iFEACEnkm1dNXsWUx34R3c0vamHrPxvfkyI1FlEUen8D1h
uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZUgvXBL046JhG
eM17+crgUyQeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFtgAsc9rk+
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1
0X90J7cC1KOyhlEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlz3+6xBIm/qe
bFyLUnf4WoiuOplAaJhK9pRY+XEnGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr
KVHUq1TZD7cvMnnNKEELTUcKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5Un5zwqkwwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqRt9X42yF1PIEVRneBRJ3HDezAgJrNh0GQWRQkhIx
```

gz6/cTR+ekr5TptVszS9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAxNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMucIINtH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+JIhZwdD8Mx2K+LVVVu/aWkNrfMuNwyDUciSI4D5QHa8T+F8fgN4OTpwYjirzel
5yoICMr9hVcbzDNv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXRYtVvYInEhLVrJ0aV6zHFoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
t/MOdjXr1fjF4pThEi5u6WsuRdFwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
2iHiOKIpQqLbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QuKE+I
R6jA0frUNt2jhiGG/F8RceXzohaaC/Cx7LUCUFWc0n7z32C9/Dtj7I1PMOacdZzz
bjJzRKO/ZDv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXPO0EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIACQUCWrVJqwIbDAAKCRC86dmkLVF4T+ZdD/9x/8APzgNJF3o3STrF
jvnV1ycyhWYGAeBJiu7wjsNWwzMFOv15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
X7DR0JszaH9wrYTxZLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt
Rwe/uwdibI0CagEzyX+2D3kTOlHO5XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZnl3LSmZyE0EQehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmcliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBWDhBPVPrruLIaenTtZEOsPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkfH6+6rbi1ujxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUUW
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcW+jsY9IhbEzqN5yQYGi4pVmDkY5vu
lXbJnbqPKpRXgM9BecV9AMbPgbDq/5LnHJJXg+G8YQOgp4lR/hC1TEFdIp5wM8AK
CWsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAr77kj7j2QR2SZeOSlFBvV7oS
mFeSNnz9xZssqrsm6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0mOmptr+xLLsKcaP7ogIXsyZnrEAEsvW8PnfayoiPCdc3cMCR/lTnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4blD32WqvyxESo7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSGm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTssoG+3OOcGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikAOlvE3/wgMqCXscbycbLjLD/bXIuFWo3rzoezeXjgi/DJx
jKBAyBTYO5nMcth1O9oaFd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvf
KeM1rO8Jm3iRAc5a/D0AEQEAAYkEPgQYAQIACQUCWrVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bXO1dQR4tKhhQ
P0LRqy6zlBY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOclTb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTfloS
Of4inq71kjmGK+TlzQ6mUMQUg228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bhIIbQyEUB9gKCmUFaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphUl7T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLrWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA0OQxG
AHMOdJQQvBsmqMvuAdjkDWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vSjKpCNg3EIJSgqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8MelN9qr5KcKQLmfdfBNEyyceBhyVl0MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRyKToTjwOb73CGkBZZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivhoWwzjpfQKhwcU9lABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBXlR
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMii+mWra23EwjChaxpvjjcUH
5ilLc5Zq781aCYRygYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAX7kPMD3Lof4KldD
Q8ppQriUvxVo+4nPV6rpTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCEl7nbFrm0vIiA75Sa8KnywTDsyZsu3XcOcf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWHOis6bq8l9RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYlNr3lydh+dFHIEkH53HzQe6l88HEic
+0jVnLkCDQRa55wJARAAyLya2Lx6gyoWoJN1a6740q3o8e9d4KggQOfGMTCflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNJyjMNT1gAgrhPNB4HtXBXum2wS57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOIzQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSvOUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJIgjNvJscuUWb
+QEqJIRleJRhrO+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIxTn9xUc0UMKFeggNAFys1K+TDTm3

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

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

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

Import the Amazon ECS PGP public key with the following command.

```
gpg --import <public_key_filename>
```

c. Download the ECS container agent signature. ECS container agent 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.

```
curl -o ecs-agent.asc https://s3.amazonaws.com/amazon-ecs-agent-us-east-1/ecs-agent-latest.tar.asc
```

d. Verify the signature.

```
gpg --verify ecs-agent.asc ./ecs-agent.tar
```

Expected output:

```
gpg: Signature made Wed 16 May 2018 08:21:06 PM UTC using RSA key ID 710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
```

```
Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

Note
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.

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 cannot access IAM role credentials from the container instance profile and enforces that tasks use only the IAM task role credentials. For more information, see IAM Roles for Tasks (p. 489).

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. 264).

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. 219).

To install the latest version of the Amazon ECS container agent on another operating system, see Installing the Amazon ECS Container Agent (p. 250). The table in Amazon ECS-optimized Amazon Linux 2 AMI Versions (p. 198) 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. 261) 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. 264) 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.

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
20200603	`1.40.0`	`19.03.6-ce`	`1.40.0-1`
20200430	`1.39.0`	`19.03.6-ce`	`1.39.0-1`
20200402	`1.39.0`	`18.09.9-ce`	`1.39.0-1`
20200319	`1.38.0`	`18.09.9-ce`	`1.38.0-1`
20200218	`1.37.0`	`18.09.9-ce`	`1.37.0-2`
20200205	`1.36.2`	`18.09.9-ce`	`1.36.2-1`
20200115	`1.36.1`	`18.09.9-ce`	`1.36.1-1`
20200108	`1.36.0`	`18.09.9-ce`	`1.36.0-1`
20191212	`1.35.0`	`18.09.9-ce`	`1.35.0-1`
20191114	`1.33.0`	`18.06.1-ce`	`1.33.0-1`
20191031	`1.32.1`	`18.06.1-ce`	`1.32.1-1`
20191014	`1.32.0`	`18.06.1-ce`	`1.32.0-1`
20190925	`1.32.0`	`18.06.1-ce`	`1.32.0-1`
20190913	`1.31.0`	`18.06.1-ce`	`1.31.0-1`
20190815	`1.30.0`	`18.06.1-ce`	`1.30.0-1`
20190709	`1.29.1`	`18.06.1-ce`	`1.29.1-1`
20190614	`1.29.0`	`18.06.1-ce`	`1.29.0-1`
20190607	`1.29.0`	`18.06.1-ce`	`1.29.0-1`
20190603	`1.28.1`	`18.06.1-ce`	`1.28.1-2`
20190510	`1.28.0`	`18.06.1-ce`	`1.28.0-1`
20190402	`1.27.0`	`18.06.1-ce`	`1.27.0-1`
20190301	`1.26.0`	`18.06.1-ce`	`1.26.0-1`
20190215	`1.25.3`	`18.06.1-ce`	`1.25.3-1`
20190204	`1.25.2`	`18.06.1-ce`	`1.25.2-1`
20190127	`1.25.1`	`18.06.1-ce`	`1.25.1-1`

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20190118	1.25.0	18.06.1-ce	1.25.0-1
20190107	1.24.0	18.06.1-ce	1.24.0-1
20181112	1.22.0	18.06.1-ce	1.22.0-1
20181016	1.20.3	18.06.1-ce	1.21.0-1

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. 190).

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. 264) 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.

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2018.03.20200603	1.40.0	19.03.6-ce	1.40.0-1
2018.03.20200430	1.39.0	19.03.6-ce	1.39.0-1
2018.03.20200402	1.39.0	18.09.9-ce	1.39.0-1
2018.03.20200319	1.38.0	18.09.9-ce	1.38.0-1
2018.03.20200218	1.37.0	18.09.9-ce	1.37.0-2
2018.03.20200205	1.36.2	18.09.9-ce	1.36.2-1
2018.03.20200115	1.36.1	18.09.9-ce	1.36.1-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20191212	1.35.0	18.09.9-ce	1.35.0-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
2018.03.20191114	`1.33.0`	`18.06.1-ce`	`1.33.0-1`
2018.03.20191031	`1.32.1`	`18.06.1-ce`	`1.32.1-1`
2018.03.20191016	`1.32.0`	`18.06.1-ce`	`1.32.0-1`
2018.03.20191014	`1.32.0`	`18.06.1-ce`	`1.32.0-1`
2018.03.y	`1.32.0`	`18.06.1-ce`	`1.32.0-1`
2018.03.x	`1.31.0`	`18.06.1-ce`	`1.31.0-1`
2018.03.w	`1.30.0`	`18.06.1-ce`	`1.30.0-1`
2018.03.v	`1.29.1`	`18.06.1-ce`	`1.29.1-1`
2018.03.u	`1.29.0`	`18.06.1-ce`	`1.29.0-1`
2018.03.t	`1.29.0`	`18.06.1-ce`	`1.29.0-1`
2018.03.s	`1.28.1`	`18.06.1-ce`	`1.28.1-2`
2018.03.q	`1.28.0`	`18.06.1-ce`	`1.28.0-1`
2018.03.p	`1.27.0`	`18.06.1-ce`	`1.27.0-1`
2018.03.o	`1.26.0`	`18.06.1-ce`	`1.26.0-1`
2018.03.n	`1.25.3`	`18.06.1-ce`	`1.25.3-1`
2018.03.m	`1.25.2`	`18.06.1-ce`	`1.25.2-1`
2018.03.l	`1.25.1`	`18.06.1-ce`	`1.25.1-1`
2018.03.k	`1.25.0`	`18.06.1-ce`	`1.25.0-1`
2018.03.j	`1.24.0`	`18.06.1-ce`	`1.24.0-1`
2018.03.i	`1.22.0`	`18.06.1-ce`	`1.22.0-1`
2018.03.h	`1.21.0`	`18.06.1-ce`	`1.21.0-1`
2018.03.g	`1.20.3`	`18.06.1-ce`	`1.20.3-1`
2018.03.f	`1.20.2`	`18.06.1-ce`	`1.20.2-1`
2018.03.e	`1.20.1`	`18.03.1-ce`	`1.20.1-1`
2018.03.d	`1.20.0`	`18.03.1-ce`	`1.20.0-1`
2018.03.c	`1.19.1`	`18.03.1-ce`	`1.19.1-1`
2018.03.b	`1.19.0`	`18.03.1-ce`	`1.19.0-1`
2018.03.a	`1.18.0`	`17.12.1-ce`	`1.18.0-1`
2017.09.l	`1.17.3`	`17.12.1-ce`	`1.17.3-1`
2017.09.k	`1.17.2`	`17.12.0-ce`	`1.17.2-1`

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
2017.09.j	`1.17.2`	`17.12.0-ce`	`1.17.2-1`
2017.09.i	`1.17.1`	`17.09.1-ce`	`1.17.1-1`
2017.09.h	`1.17.0`	`17.09.1-ce`	`1.17.0-2`
2017.09.g	`1.16.2`	`17.09.1-ce`	`1.16.2-1`
2017.09.f	`1.16.1`	`17.06.2-ce`	`1.16.1-1`
2017.09.e	`1.16.1`	`17.06.2-ce`	`1.16.1-1`
2017.09.d	`1.16.0`	`17.06.2-ce`	`1.16.0-1`
2017.09.c	`1.15.2`	`17.06.2-ce`	`1.15.1-1`
2017.09.b	`1.15.1`	`17.06.2-ce`	`1.15.1-1`
2017.09.a	`1.15.0`	`17.06.2-ce`	`1.15.0-4`
2017.03.g	`1.14.5`	`17.03.2-ce`	`1.14.5-1`
2017.03.f	`1.14.4`	`17.03.2-ce`	`1.14.4-1`
2017.03.e	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2017.03.d	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2017.03.c	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2017.03.b	`1.14.3`	`17.03.1-ce`	`1.14.3-1`
2016.09.g	`1.14.1`	`1.12.6`	`1.14.1-1`
2016.09.f	`1.14.0`	`1.12.6`	`1.14.0-2`
2016.09.e	`1.14.0`	`1.12.6`	`1.14.0-1`
2016.09.d	`1.13.1`	`1.12.6`	`1.13.1-2`
2016.09.c	`1.13.1`	`1.11.2`	`1.13.1-1`
2016.09.b	`1.13.1`	`1.11.2`	`1.13.1-1`
2016.09.a	`1.13.0`	`1.11.2`	`1.13.0-1`
2016.03.j	`1.13.0`	`1.11.2`	`1.13.0-1`
2016.03.i	`1.12.2`	`1.11.2`	`1.12.2-1`
2016.03.h	`1.12.1`	`1.11.2`	`1.12.1-1`
2016.03.g	`1.12.0`	`1.11.2`	`1.12.0-1`
2016.03.f	`1.11.1`	`1.11.2`	`1.11.1-1`
2016.03.e	`1.11.0`	`1.11.2`	`1.11.0-1`
2016.03.d	`1.10.0`	`1.11.1`	`1.10.0-1`

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	`ecs-init` version
2016.03.c	`1.10.0`	`1.11.1`	`1.10.0-1`
2016.03.b	`1.9.0`	`1.9.1`	`1.9.0-1`
2016.03.a	`1.8.2`	`1.9.1`	`1.8.2-1`
2015.09.g	`1.8.1`	`1.9.1`	`1.8.1-1`
2015.09.f	`1.8.0`	`1.9.1`	`1.8.0-1`
2015.09.e	`1.7.1`	`1.9.1`	`1.7.1-1`
2015.09.d	`1.7.1`	`1.9.1`	`1.7.1-1`
2015.09.c	`1.7.0`	`1.7.1`	`1.7.0-1`
2015.09.b	`1.6.0`	`1.7.1`	`1.6.0-1`
2015.09.a	`1.5.0`	`1.7.1`	`1.5.0-1`
2015.03.g	`1.4.0`	`1.7.1`	`1.4.0-2`
2015.03.f	`1.4.0`	`1.6.2`	`1.4.0-1`
2015.03.e	`1.3.1`	`1.6.2`	`1.3.1-1`
2015.03.d	`1.2.1`	`1.6.2`	`1.2.0-2`
2015.03.c	`1.2.0`	`1.6.2`	`1.2.0-1`
2015.03.b	`1.1.0`	`1.6.0`	`1.0-3`
2015.03.a	`1.0.0`	`1.5.0`	`1.0-1`

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. 190).

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

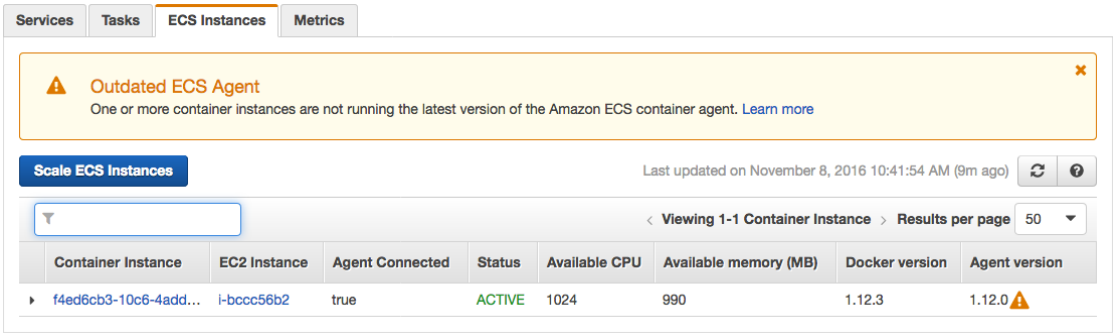
- Checking Your Amazon ECS Container Agent Version (p. 265)
- Updating the Amazon ECS Container Agent on an Amazon ECS-optimized AMI (p. 266)
- Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-Optimized AMIs) (p. 268)

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 the container instance does not contain the latest version of the container agent, the console alerts you with a message and flags the outdated agent version.



   If your agent version is outdated, you can update your container agent with the following procedures:

   - If your container instance is running an Amazon ECS-optimized AMI, see Updating the Amazon ECS Container Agent on an Amazon ECS-optimized AMI (p. 266).
   - 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. 268).

   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. 219).

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. 309).

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
```

Note
The introspection API added `Version` information in the version v1.0.0 of the Amazon ECS container agent. If `Version` is not present when querying the introspection API, or the introspection API is not present in your agent at all, then the version you are running is v0.0.3 or earlier. You should update your version.

Updating the Amazon ECS Container Agent on an Amazon ECS-optimized AMI

If you are using an Amazon ECS-optimized AMI, you have several options to get the latest version of the Amazon ECS container agent (shown in order of recommendation):

- Terminate 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. 190).
- 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. 266).
- 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. 267).

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. 236).
2. Update the `ecs-init` package with the following command.

```
[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

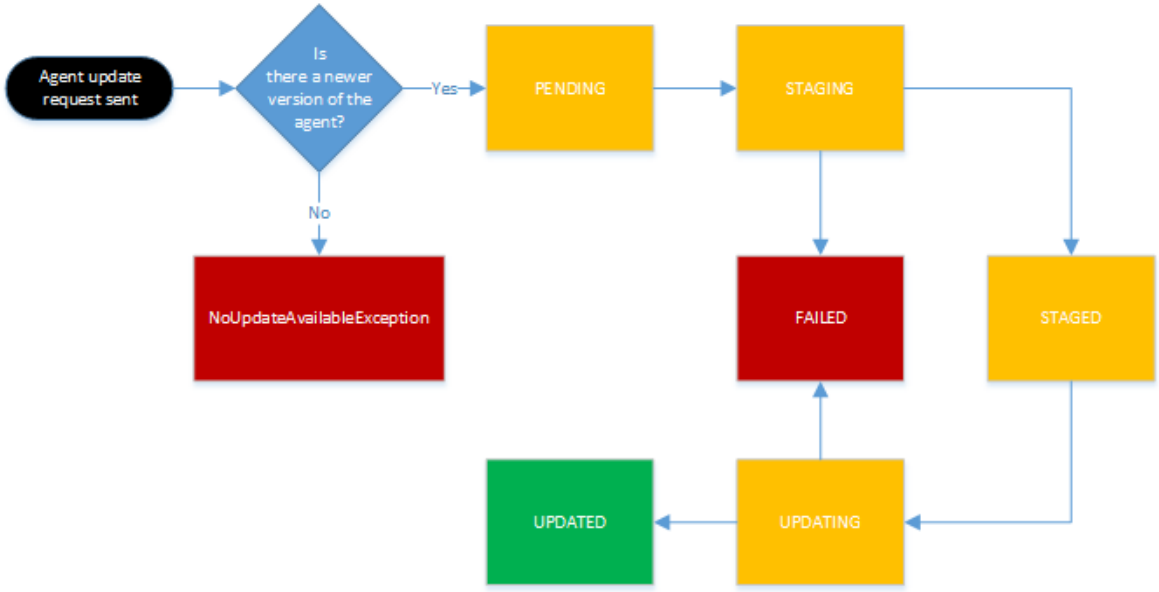
This update process is only supported on Linux variants of the Amazon ECS-optimized AMI. For container instances that are running other operating systems, see Manually Updating the Amazon ECS Container Agent (for Non-Amazon ECS-Optimized AMIs) (p. 268).

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. 219).

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.



The stages in the update process shown above are as follows:

`PENDING`

An agent update is available, and the update process has started.

`STAGING`

The agent has begun downloading the agent update. If the agent cannot download the update, or if the contents of the update are incorrect or corrupted, then the agent sends a notification of the failure and the update transitions to the `FAILED` state.

`STAGED`

The agent download has completed and the agent contents have been verified.

`UPDATING`

The `ecs-init` service is restarted and it picks up the new agent version. If the agent is for some reason unable to restart, the update transitions to the `FAILED` state; otherwise, the agent signals Amazon ECS that the update is complete.

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.

```
ubuntu:~$ docker inspect ecs-agent | grep ECS_DATADIR
```

Output:

```
"ECS_DATADIR=/data",
```

Important
If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before updating your agent. Newer

agents with the `ECS_DATADIR` environment variable save their state and you can update them while tasks are running without issues.

3. Stop the Amazon ECS container agent.

```
ubuntu:~$ docker stop ecs-agent
```

4. Delete the agent container.

```
ubuntu:~$ docker rm ecs-agent
```

5. Ensure that the `/etc/ecs` directory and the Amazon ECS container agent configuration file exist at `/etc/ecs/ecs.config`.

```
ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
```

6. Edit the `/etc/ecs/ecs.config` file and ensure that it contains at least the following variable declarations. If you do not want your container instance to register with the default cluster, specify your cluster name as the value for `ECS_CLUSTER`.

```
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
```

For more information about these and other agent runtime options, see Amazon ECS Container Agent Configuration (p. 270).

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. 283) and Private Registry Authentication for Tasks (p. 160).

7. Pull the latest Amazon ECS container agent image from Docker Hub.

```
ubuntu:~$ docker pull amazon/amazon-ecs-agent:latest
```

Output:

```
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
511136ea3c5a: Download complete
9950b5d678a1: Download complete
c48ddcf21b63: Download complete
Status: Image is up to date for amazon/amazon-ecs-agent:latest
```

8. Run the latest Amazon ECS container agent on your container instance.

Note
Use Docker restart policies or a process manager (such as **upstart** or **systemd**) to treat the container agent as a service or a daemon and ensure that it is restarted after exiting. For more information, see Automatically start containers and Restart policies 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 (p. 226).

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. 270).

Important
Operating systems with SELinux enabled require the `--privileged` option in your **docker run** command. In addition, for SELinux-enabled container instances, we recommend that you add the `:Z` option to the `/log` and `/data` volume mounts. However, the host mounts for these volumes must exist before you run the command or you receive a `no such file or directory` error. Take the following action if you experience difficulty running the Amazon ECS agent on an SELinux-enabled container instance:

- Create the host volume mount points on your container instance.

```
ubuntu:~$ sudo mkdir -p /var/log/ecs /var/lib/ecs/data
```

- Add the `--privileged` option to the **docker run** command below.
- Append the `:Z` option to the `/log` and `/data` container volume mounts (for example, `--volume=/var/log/ecs/:/log:Z`) to the **docker run** command below.

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run:/var/run \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
--volume=/etc/ecs:/etc/ecs \
--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. 223).

If you are manually starting the Amazon ECS container agent (for non Amazon ECS-optimized AMIs), you can use these environment variables in the **docker run** command that you use to start the agent. Use these variables with the syntax `--env=`*`VARIABLE_NAME`*`=`*`VARIABLE_VALUE`*. For sensitive information, such as authentication credentials for private repositories, you should store your agent environment variables in a file and pass them all at one time with the `--env-file` *`path_to_env_file`* option.

Topics

- Available Parameters (p. 271)
- Storing Container Instance Configuration in Amazon S3 (p. 283)

Available Parameters

The following are the available Amazon ECS container agent configuration parameters. There are undocumented variables that the agent uses internally that may be visible but that are not intended for customer use. For more information, see Amazon ECS Container Agent on GitHub.

`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, then 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. 284).

`ECS_ENGINE_AUTH_DATA`

Example values:

- `ECS_ENGINE_AUTH_TYPE=dockercfg`: `{"https://index.docker.io/v1/": {"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}`

- `ECS_ENGINE_AUTH_TYPE=docker`: `{"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. 284).

`AWS_DEFAULT_REGION`

Example values: `us-east-1`

Default value on Linux: Taken from Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon 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 Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon 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 Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon EC2 instance metadata.

The secret key used by the agent for all calls.

`AWS_SESSION_TOKEN`

Default value on Linux: Taken from Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon 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_LOGFILE`

Example values: `/ecs-agent.log`

Default value on Linux: Null

Default value on Windows: Null

The location where agent logs should be written. If you are running the agent via `ecs-init`, which is the default method when using the Amazon ECS-optimized AMI, the in-container path will be `/log` and `ecs-init` mounts that out to `/var/log/ecs/` on the host.

`ECS_LOGLEVEL`

Example values: `crit`, `error`, `warn`, `info`, `debug`

Default value on Linux: `info`

Default value on Windows: `info`

The level of detail to log.

`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 Amazon ECS agent updates when they are requested.

`ECS_DISABLE_METRICS`

Example values: `true` | `false`

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 Amazon ECS tasks. For more information, see Container Instance Memory Management (p. 240).

`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. 143). 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 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 to wait from when a task is stopped before its containers are forcefully stopped 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 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 Amazon ECS agent to connect to the internet. For example, this proxy will be used 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 Amazon EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see HTTP Proxy Configuration (p. 311).

`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 Amazon EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see HTTP Proxy Configuration (p. 311).

`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. 489).

`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. 489).

`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. 287).

`ECS_IMAGE_CLEANUP_INTERVAL`

Example values: `30m`

Default value on Linux: `30m`

Default value on Windows: `30m`

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.

`NON_ECS_IMAGE_MINIMUM_CLEANUP_AGE`

Example values: `30m`

Default value on Linux: `1h`

Default value on Windows: `1h`

The minimum time interval between when a non-Amazon ECS image is created 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

Default value on Windows: Null

A list of custom attributes, in JSON format, 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 through 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. 324).

For information about custom attributes to use, see Attributes (p. 323).

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, such as when 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. 701).

`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 when the Amazon ECS agent is running as a container. We do not use this value in Windows because the Amazon 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 Amazon ECS agent. This is the path that is 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`, Amazon 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 `autoprovision` 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. 126).

`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. 410).

`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. 410).

`ECS_ENABLE_UNTRACKED_IMAGE_CLEANUP`

Example values: `true` | `false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to allow the Amazon 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 Amazon 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 Amazon 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. 121).

`ECS_ENABLE_SPOT_INSTANCE_DRAINING`

Example values: `true`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to enable Spot Instance draining for the container instance. When true, if the container instance receives a Spot interruption notice, then the agent sets the instance status to `DRAINING`,

which gracefully shuts down and replaces all tasks running on the instance that are part of a service. It is recommended that this be set to true when using Spot Instances. For more information, see Container Instance Draining (p. 239).

`ECS_LOG_ROLLOVER_TYPE`

Example values: `size`, `hourly`

Default value on Linux: `hourly`

Default value on Windows: `hourly`

Determines whether the container agent log file will be rotated hourly or based on size. By default, the agent log file is rotated each hour.

`ECS_LOG_OUTPUT_FORMAT`

Example values: `logfmt`, `json`

Default value on Linux: `logfmt`

Default value on Windows: `logfmt`

Determines the log output format. When the `json` format is used, each line in the log will be a structured JSON map.

`ECS_LOG_MAX_FILE_SIZE_MB`

Example values: `10`

Default value on Linux: `10`

Default value on Windows: `10`

When the `ECS_LOG_ROLLOVER_TYPE` variable is set to `size`, this variable determines the maximum size (in MB) of the log file before it is rotated. If the rollover type is set to `hourly`, then this variable is ignored.

`ECS_LOG_MAX_ROLL_COUNT`

Example values: `24`

Default value on Linux: `24`

Default value on Windows: `24`

Determines the number of rotated log files to keep. Older log files are deleted once this limit is reached.

`ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE`

Example values: `true`

Default value on Linux: *see description below*

Default value on Windows: `false`

Whether to enable the `awslogs` log driver to authenticate using the task execution IAM role. This parameter must be `true` to use the `awslogs` log driver in a task that also has a task execution IAM role specified. When using the Amazon ECS-optimized AMI with version 1.16.0-1 or later of the `ecs-init` package, the default value of `true` is used. When using older versions of the `ecs-init` package, the default value of `false` is used.

Storing Container Instance Configuration in Amazon S3

Amazon ECS container agent configuration is controlled with the environment variables described in the previous section. Linux variants of the Amazon ECS-optimized AMI look for these variables in `/etc/ecs/ecs.config` when the container agent starts and configure the agent accordingly. 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 would allow them to show up in a `.bash_history` file.

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance configuration at launch. You can store a copy of your `ecs.config` file in a private bucket. You can then use Amazon EC2 user data to install the AWS CLI and copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

To 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. 486).
3. Under **Managed Policies**, choose **Attach Policy**.
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. 270) using the following format. This example configures private registry authentication. For more information, see Private Registry Authentication for Tasks (p. 160).

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","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. 219). In Step 6.g (p. 220), use the following example script that installs the AWS CLI and copies your configuration file to `/etc/ecs/ecs.config`.

```
#!/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. 160).

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. 283).

Authentication Formats

There are two available formats for private registry authentication, `dockercfg` and `docker`.

dockercfg Authentication Format

The `dockercfg` format uses the authentication information stored in the configuration file that is created when you run the **docker login** command. You can create this file by running **docker login** on your local system and entering your registry user name, password, and email address. You can also log in to a container instance and run the command there. Depending on your Docker version, this file is saved as either `~/.dockercfg` or `~/.docker/config.json`.

```
cat ~/.docker/config.json
```

Output:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "zq212MzEXAMPLE7o6T25Dk0i"
```

```
    }
  }
}
```

Important
Newer versions of Docker create a configuration file as shown above with an outer `auths` object. The Amazon ECS agent only supports `dockercfg` authentication data that is in the below format, without the `auths` object. If you have the **jq** utility installed, you can extract this data with the following command: **cat ~/.docker/config.json | jq .auths**

```
cat ~/.docker/config.json | jq .auths
```

Output:

```
{
  "https://index.docker.io/v1/": {
    "auth": "zq212MzEXAMPLE7o6T25Dk0i",
    "email": "email@example.com"
  }
}
```

In the above example, the following environment variables should be added to the environment variable file (`/etc/ecs/ecs.config` for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at runtime. If you are not using an Amazon ECS-optimized AMI and you are starting the agent manually with **docker run**, specify the environment variable file with the `--env-file` *`path_to_env_file`* option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=dockercfg
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example-01.com"},"repo.example-02.com":
{"auth":"fQ172MzEXAMPLEoF7225DU0j","email":"email@example-02.com"}}
```

docker Authentication Format

The `docker` format uses a JSON representation of the registry server that the agent should authenticate with. It also includes the authentication parameters required by that registry (such as user name, password, and the email address for that account). For a Docker Hub account, the JSON representation looks like the following:

```
{
  "https://index.docker.io/v1/": {
    "username": "my_name",
    "password": "my_password",
    "email": "email@example.com"
  }
}
```

In this example, the following environment variables should be added to the environment variable file (`/etc/ecs/ecs.config` for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at runtime. If you are not using an Amazon ECS-optimized AMI, and you are starting the agent manually

with **docker run**, specify the environment variable file with the `--env-file` *path_to_env_file* option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
{"username":"my_name","password":"my_password","email":"email@example.com"}}
```

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":
{"username":"my_name","password":"my_password","email":"email@example-01.com"},"repo.example-02.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:

   ```
   sudo vi /etc/ecs/ecs.config
   ```

   This example authenticates a Docker Hub user account:

   ```
   ECS_ENGINE_AUTH_TYPE=docker
   ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
   {"username":"my_name","password":"my_password","email":"email@example.com"}}
   ```

3. Check to see if your agent uses the `ECS_DATADIR` environment variable to save its state:

   ```
   docker inspect ecs-agent | grep ECS_DATADIR
   ```

   Output:

   ```
   "ECS_DATADIR=/data",
   ```

   Important
   If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before stopping the agent. Newer agents with the `ECS_DATADIR` environment variable save their state and you can stop and start them while tasks are running without issues. For more information, see Updating the Amazon ECS Container Agent (p. 264).

4. Stop the `ecs` service:

   ```
   sudo stop ecs
   ```

   Output:

   ```
   ecs stop/waiting
   ```

5. Restart the `ecs` service.

   - For the Amazon ECS-optimized Amazon Linux 2 AMI:

   ```
   sudo systemctl restart ecs
   ```

   - For the Amazon ECS-optimized Amazon Linux AMI:

   ```
   sudo stop ecs && sudo start ecs
   ```

6. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see the section called "Amazon ECS Container Agent Introspection" (p. 309).

   ```
   curl http://localhost:51678/v1/metadata
   ```

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. 270).

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. 264).

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. 270).

`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 your containers or the host container instance. By enabling this feature, you can query the information about a task, container, and container instance from within the container or the host container instance. The metadata file is created on the host instance and mounted in the container as a Docker volume.

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. 287).

Topics

- Enabling Container Metadata (p. 288)
- Container Metadata File Locations (p. 289)
- Container Metadata File Format (p. 289)

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. 270).

If the `ECS_ENABLE_CONTAINER_METADATA` is set to `true` when the agent starts, metadata files are created for any containers created from that point forward. The Amazon ECS container agent cannot create metadata files for containers that were created before the `ECS_ENABLE_CONTAINER_METADATA` container agent variable was set to `true`. To ensure that all containers receive metadata files, you should set this agent variable at container instance launch. The following is an example user data script that will set this variable as well as register your container instance with your cluster.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_ENABLE_CONTAINER_METADATA=true
EOF
```

Container Metadata File Locations

By default, the container metadata file is written to the following host and container paths.

- **For Linux instances:**
  - Host path: `/var/lib/ecs/data/metadata/`*`cluster_name`*`/`*`task_id`*`/`*`container_name`*`/ecs-container-metadata.json`

    Note
    The Linux host path assumes that the default data directory mount path (`/var/lib/ecs/data`) is used when the agent is started. If you are not using an Amazon ECS-optimized AMI (or the `ecs-init` package to start and maintain the container agent), be sure to set the `ECS_HOST_DATA_DIR` agent configuration variable to the host path where the container agent's state file is located. For more information, see Amazon ECS Container Agent Configuration (p. 270).
  - 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.

`TaskDefinitionFamily`

The name of the task definition family the container is using.

`TaskDefinitionRevision`

The task definition revision the container is using.

`ContainerID`

The Docker container ID (and not the Amazon ECS container ID) for the container.

`ContainerName`

The container name from the Amazon ECS task definition for the container.

`DockerContainerName`

The container name that the Docker daemon uses for the container (for example, the name that shows up in **docker ps** command output).

`ImageID`

The SHA digest for the Docker image used to start the container.

`ImageName`

The image name and tag for the Docker image used to start the container.

`PortMappings`

Any port mappings associated with the container.

`ContainerPort`

The port on the container that is exposed.

`HostPort`

The port on the host container instance that is exposed.

`BindIp`

The bind IP address that is assigned to the container by Docker. This IP address is only applied with the `bridge` network mode, and it is only accessible from the container instance.

`Protocol`

The network protocol used for the port mapping.

`Networks`

The network mode and IP address for the container.

`NetworkMode`

The network mode for the task to which the container belongs.

`IPv4Addresses`

The IP addresses associated with the container.

Important
If your task is using the `awsvpc` network mode, the IP address of the container will not be returned. In this case, you can retrieve the IP address by reading the /etc/hosts file with the following command:

```
cat /etc/hosts | tail -1 | awk {'print $1'}
```

`MetadataFileStatus`

The status of the metadata file. When the status is `READY`, the metadata file is current and complete. If the file is not ready yet (for example, the moment the task is started), a truncated version of the file format is available. To avoid a likely race condition where the container has started, but the metadata has not yet been written, you can parse the metadata file and wait for this parameter to be set to `READY` before depending on the metadata. This is usually available in less than 1 second from when the container starts.

`AvailabilityZone`

The Availability Zone the host container instance resides in.

`HostPrivateIPv4Address`

The private IP address for the task the container belongs to.

`HostPublicIPv4Address`

The public IP address for the task the container belongs to.

Example Amazon ECS container metadata file (`READY`)

The following example shows a container metadata file in the `READY` status.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/
default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/2b88376d-aba3-4950-9ddf-
bcb0f388a40c",
    "TaskDefinitionFamily": "console-sample-app-static",
    "TaskDefinitionRevision": "1",
    "ContainerID": "aec2557997f4eed9b280c2efd7afccdcedfda4ac399f7480cae870cfc7e163fd",
    "ContainerName": "simple-app",
    "DockerContainerName": "/ecs-console-sample-app-static-1-simple-app-
e4e8e495e8baa5de1a00",
    "ImageID": "sha256:2ae34abc2ed0a22e280d17e13f9c01aaf725688b09b7a1525d1a2750e2c0d1de",
    "ImageName": "httpd:2.4",
    "PortMappings": [
        {
            "ContainerPort": 80,
            "HostPort": 80,
            "BindIp": "0.0.0.0",
            "Protocol": "tcp"
        }
    ],
    "Networks": [
        {
            "NetworkMode": "bridge",
            "IPv4Addresses": [
                "192.0.2.0"
```

```
            ]
        }
    ],
    "MetadataFileStatus": "READY",
    "AvailabilityZone": "us-east-1b",
    "HostPrivateIPv4Address": "192.0.2.0",
    "HostPublicIPv4Address": "203.0.113.0"
}
```

Example Incomplete Amazon ECS container metadata file (not yet `READY`)

The following example shows a container metadata file that has not yet reached the `READY` status. The information in the file is limited to a few parameters that are known from the task definition. The container metadata file should be ready within 1 second after the container starts.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/
default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/
d90675f8-1a98-444b-805b-3d9cabb6fcd4",
    "ContainerName": "metadata"
}
```

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 4 – Available for tasks that use the Fargate launch type on platform version 1.4.0 or later and tasks that use the EC2 launch type and are launched on Amazon EC2 infrastructure running at least version 1.39.0 of the Amazon ECS container agent. For more information, see Task Metadata Endpoint version 4 (p. 292).
- Task metadata endpoint version 3 – Available for tasks that use the Fargate launch type on platform version 1.3.0 or later and 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. 300).
- Task metadata endpoint version 2 – Available for tasks that use the Fargate launch type on platform version 1.1.0 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. 305).

Topics

- Task Metadata Endpoint version 4 (p. 292)
- Task Metadata Endpoint version 3 (p. 300)
- Task Metadata Endpoint version 2 (p. 305)

Task Metadata Endpoint version 4

Beginning with Amazon ECS container agent version 1.39.0 and Fargate platform version 1.4.0, an environment variable named `ECS_CONTAINER_METADATA_URI_V4` is injected into each container in a task. When you query the task metadata version 4 endpoint, various task metadata and Docker stats are available to tasks.

The task metadata version 4 endpoint functions like version 3 but provides additional network metadata for your containers and tasks. For tasks that use the `awsvpc` network mode, additional network metrics are available when querying the `/stats` endpoints.

Important
To avoid the need to create new task metadata endpoint versions in the future, additional metadata may be added to the version 4 output. We will not remove any existing metadata or change the metadata field names.

The following additional network metadata is included when querying the task metadata version 4 endpoint:

- `AttachmentIndex`
- `IPV4SubnetCIDRBlock`
- `MACAddress`
- `PrivateDNSName`
- `SubnetGatewayIPV4Address`
- `DomainNameServers` (Fargate tasks only)
- `DomainNameSearchList` (Fargate tasks only)

Enabling Task Metadata

The task metadata endpoint version 4 feature is enabled by default for tasks that use the Fargate launch type on platform version 1.4.0 or later and tasks that use the EC2 launch type and are launched on Amazon EC2 infrastructure running at least version 1.39.0 of the Amazon ECS container agent. For more information, see Amazon ECS Container Agent Versions (p. 259).

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. 264).

Important
For tasks using the Fargate launch type and platform versions prior to 1.4.0, the task metadata version 3 and 2 endpoint are supported. For more information, see Task Metadata Endpoint version 3 (p. 300) or Task Metadata Endpoint version 2 (p. 305).

Task Metadata Endpoint version 4 Paths

The following task metadata endpoints are available to containers:

`${ECS_CONTAINER_METADATA_URI_V4}`

This path returns metadata JSON for the container.

`${ECS_CONTAINER_METADATA_URI_V4}/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. 294).

`${ECS_CONTAINER_METADATA_URI_V4}/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_V4}/task/stats`

This path returns Docker stats JSON for all of the containers associated with the task. This can be used by sidecar containers to extract network metrics. For more information about each of the returned stats, see ContainerStats in the Docker API documentation.

Task Metadata JSON Response

The following information is returned from the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON response.

`Cluster`

The full Amazon Resource Name (ARN) of 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.

`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 timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

Note
The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later.

`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.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Examples

The following examples show sample outputs from the task metadata endpoints.

Example Container Metadata Response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}` endpoint you are returned only metadata about the container itself. The following is an example output.

```
{
    "DockerId": "c7a6b9b237934e9999f319ea3ccc9da4query-metadata",
    "Name": "query-metadata",
    "DockerName": "query-metadata",
    "Image": "mreferre/eksutils",
    "ImageID": "sha256:1b146e73f801617610dcb00441c6423e7c85a7583dd4a65ed1be03cb0e123311",
    "Labels": {
        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:&ExampleAWSAccountNo1;:cluster/
default",
        "com.amazonaws.ecs.container-name": "query-metadata",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:&ExampleAWSAccountNo1;:task/
default/c7a6b9b237934e9999f319ea3ccc9da4",
        "com.amazonaws.ecs.task-definition-family": "query-metadata",
        "com.amazonaws.ecs.task-definition-version": "3"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 2
    },
    "CreatedAt": "2020-03-26T22:11:23.62831313Z",
    "StartedAt": "2020-03-26T22:11:23.62831313Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.0.61"
            ],
            "AttachmentIndex": 0,
            "IPv4SubnetCIDRBlock": "10.0.0.0/24",
            "MACAddress": "0a:d2:d0:80:b6:b4",
            "DomainNameServers": [
                "10.0.0.2"
            ],
            "DomainNameSearchList": [
                "us-west-2.compute.internal"
            ],
            "PrivateDNSName": "ip-10-0-0-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": ""
        }
    ]
}
```


Example Task Metadata Response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/task` endpoint you are returned metadata about the task the container is part of. The following is an example output.


```
{
    "Cluster": "arn:aws:ecs:us-west-2:&ExampleAWSAccountNo1;:cluster/default",
    "TaskARN": "arn:aws:ecs:us-west-2:&ExampleAWSAccountNo1;:task/default/
febee046097849aba589d4435207c04a",
    "Family": "query-metadata",
    "Revision": "7",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 0.25,
        "Memory": 512
    },
    "PullStartedAt": "2020-03-26T22:25:40.420726088Z",
    "PullStoppedAt": "2020-03-26T22:26:22.235177616Z",
```

```
    "AvailabilityZone": "us-west-2c",
    "Containers": [
        {
            "DockerId": "febee046097849aba589d4435207c04aquery-metadata",
            "Name": "query-metadata",
            "DockerName": "query-metadata",
            "Image": "mreferre/eksutils",
            "ImageID":
 "sha256:1b146e73f801617610dcb00441c6423e7c85a7583dd4a65ed1be03cb0e123311",
            "Labels": {
                "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-
west-2:&ExampleAWSAccountNo1;:cluster/default",
                "com.amazonaws.ecs.container-name": "query-metadata",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
west-2:&ExampleAWSAccountNo1;:task/default/febee046097849aba589d4435207c04a",
                "com.amazonaws.ecs.task-definition-family": "query-metadata",
                "com.amazonaws.ecs.task-definition-version": "7"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 2
            },
            "CreatedAt": "2020-03-26T22:26:24.534553758Z",
            "StartedAt": "2020-03-26T22:26:24.534553758Z",
            "Type": "NORMAL",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.0.108"
                    ],
                    "AttachmentIndex": 0,
                    "IPv4SubnetCIDRBlock": "10.0.0.0/24",
                    "MACAddress": "0a:62:17:7a:36:68",
                    "DomainNameServers": [
                        "10.0.0.2"
                    ],
                    "DomainNameSearchList": [
                        "us-west-2.compute.internal"
                    ],
                    "PrivateDNSName": "ip-10-0-0-108.us-west-2.compute.internal",
                    "SubnetGatewayIpv4Address": ""
                }
            ]
        }
    ]
}
```


Example Task Stats Response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/task/stats` endpoint you are returned network metrics about the task the container is part of. The following is an example output.


```
{
    "1823e1f6-7248-43c3-bed6-eea1fa7501a5query-metadata": {
        "read": "2020-04-06T16:12:01.090148907Z",
        "preread": "2020-04-06T16:11:56.083890951Z",
        "pids_stats": {

        },
        "blkio_stats": {
            "io_service_bytes_recursive": [
                {
```

```
                "major": 202,
                "minor": 26368,
                "op": "Read",
                "value": 3452928
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Write",
                "value": 0
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Sync",
                "value": 3452928
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Async",
                "value": 0
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Total",
                "value": 3452928
            }
        ],
        "io_serviced_recursive": [
            {
                "major": 202,
                "minor": 26368,
                "op": "Read",
                "value": 118
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Write",
                "value": 0
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Sync",
                "value": 118
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Async",
                "value": 0
            },
            {
                "major": 202,
                "minor": 26368,
                "op": "Total",
                "value": 118
            }
        ],
        "io_queue_recursive": [

        ],
        "io_service_time_recursive": [
```

```
        ],
        "io_wait_time_recursive": [

        ],
        "io_merged_recursive": [

        ],
        "io_time_recursive": [

        ],
        "sectors_recursive": [

        ]
    },
    "num_procs": 0,
    "storage_stats": {

    },
    "cpu_stats": {
        "cpu_usage": {
            "total_usage": 410557100,
            "percpu_usage": [
                410557100,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0
            ],
            "usage_in_kernelmode": 10000000,
            "usage_in_usermode": 250000000
        },
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
            "throttled_time": 0
        }
    },
    "precpu_stats": {
        "cpu_usage": {
            "total_usage": 0,
            "usage_in_kernelmode": 0,
            "usage_in_usermode": 0
        },
        "throttling_data": {
            "periods": 0,
            "throttled_periods": 0,
            "throttled_time": 0
        }
    },
    "memory_stats": {
        "usage": 4390912,
        "max_usage": 6488064,
        "stats": {
            "active_anon": 278528,
            "active_file": 344064,
```

```
                "cache": 3452928,
                "dirty": 0,
                "hierarchical_memory_limit": 536870912,
                "hierarchical_memsw_limit": 9223372036854772000,
                "inactive_anon": 0,
                "inactive_file": 3108864,
                "mapped_file": 2412544,
                "pgfault": 2800,
                "pgmajfault": 28,
                "pgpgin": 3144,
                "pgpgout": 2233,
                "rss": 278528,
                "rss_huge": 0,
                "total_active_anon": 278528,
                "total_active_file": 344064,
                "total_cache": 3452928,
                "total_dirty": 0,
                "total_inactive_anon": 0,
                "total_inactive_file": 3108864,
                "total_mapped_file": 2412544,
                "total_pgfault": 2800,
                "total_pgmajfault": 28,
                "total_pgpgin": 3144,
                "total_pgpgout": 2233,
                "total_rss": 278528,
                "total_rss_huge": 0,
                "total_unevictable": 0,
                "total_writeback": 0,
                "unevictable": 0,
                "writeback": 0
            },
            "limit": 9223372036854772000
        },
        "name": "query-metadata",
        "id": "1823e1f6-7248-43c3-bed6-eea1fa7501a5query-metadata",
        "networks": {
            "eth1": {
                "rx_bytes": 564655295,
                "rx_packets": 384960,
                "rx_errors": 0,
                "rx_dropped": 0,
                "tx_bytes": 3043269,
                "tx_packets": 54355,
                "tx_errors": 0,
                "tx_dropped": 0
            }
        }
    }
}
```


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. For tasks that use the `bridge` network mode, network metrics are available when querying the `/stats` endpoints.

Enabling Task Metadata

The task metadata endpoint version 3 feature is enabled by default for tasks that use the Fargate launch type on platform version v1.3.0 or later and tasks that use the EC2 launch type and are launched on

Amazon EC2 infrastructure running at least version 1.21.0 of the Amazon ECS container agent. For more information, see Amazon ECS Container Agent Versions (p. 259).

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. 264).

Important
For tasks using the Fargate launch type and platform versions prior to v1.3.0, the task metadata version 2 endpoint is supported. For more information, see Task Metadata Endpoint version 2 (p. 305).

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. 301).

`${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 full Amazon Resource Name (ARN) of 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.

`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 timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

Note
The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later.

`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.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Examples

The following examples show sample outputs from the task metadata endpoints.

Example Container Metadata Response

When querying the `${ECS_CONTAINER_METADATA_URI}` endpoint you are returned only metadata about the container itself. The following is an example output.

```
{
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
    "Name": "nginx-curl",
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
    "Image": "nrdlngr/nginx-curl",
    "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 512,
        "Memory": 512
    },
    "CreatedAt": "2018-02-01T20:55:10.554941919Z",
    "StartedAt": "2018-02-01T20:55:11.064236631Z",
    "Type": "NORMAL",
```

```
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.106"
            ]
        }
    ]
}
```

Example Task Metadata Response

When querying the `${ECS_CONTAINER_METADATA_URI}/task` endpoint you are returned metadata about the task the container is part of. The following is an example output.

The following JSON response is for a single-container task.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
```

```
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

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. 33).
- 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. 259).

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. 264).

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. 306).

`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 full Amazon Resource Name (ARN) of 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.

`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 timestamp for when the first container image pull began.

`PullStoppedAt`

The timestamp for when the last container image pull finished.

`AvailabilityZone`

The Availability Zone the task is in.

Note
The Availability Zone metadata is only available for Fargate tasks using platform version 1.4 or later.

`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.

`ExecutionStoppedAt`

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

Example Task Metadata Response

The following JSON response is for a single-container task.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-
f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal~ecs~pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal~ecs~pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    },
    {
      "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
      "Name": "nginx-curl",
      "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
      "Image": "nrdlngr/nginx-curl",
      "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
```

```
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Limits": {
        "CPU": 512,
        "Memory": 512
      },
      "CreatedAt": "2018-02-01T20:55:10.554941919Z",
      "StartedAt": "2018-02-01T20:55:11.064236631Z",
      "Type": "NORMAL",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
  "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
  "AvailabilityZone": "us-east-2b"
}
```

Amazon ECS 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. 486).

To view container instance metadata, log in to your container instance via SSH and run the following command. Metadata includes the container instance ID, the Amazon ECS cluster in which the container instance is registered, and the Amazon ECS container agent version information.

```
curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Output:

```
{
    "Cluster": "cluster_name",
    "ContainerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-
instance/cluster_name/container_instance_id",
    "Version": "Amazon ECS Agent - v1.30.0 (02ff320c)"
}
```

To view information about all of the tasks that are running on a container instance, log in to your container instance via SSH and run the following command:

```
curl http://localhost:51678/v1/tasks
```

Output:

```
{
  "Tasks": [
    {
      "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/example5-58ff-46c9-
ae05-543f8example",
      "DesiredStatus": "RUNNING",
      "KnownStatus": "RUNNING",
      "Family": "hello_world",
      "Version": "8",
      "Containers": [
        {
          "DockerId": "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
          "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
          "Name": "mysql"
        },
        {
          "DockerId": "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
          "DockerName": "ecs-hello_world-8-wordpress-e8bfddf9b488dff36c00",
          "Name": "wordpress"
        }
      ]
    }
  ]
}
```

You can view information for a particular task that is running on a container instance. To specify a specific task or container, append one of the following to the request:

- The task ARN (`?taskarn=`*`task_arn`*)
- The Docker ID for a container (`?dockerid=`*`docker_id`*)

To get task information with a container's Docker ID, log in to your container instance via SSH and run the following command.

> **Note**
> Amazon ECS container agents before version 1.14.2 require full Docker container IDs for the introspection API, not the short version that is shown with **docker ps**. You can get the full Docker ID for a container by running the **docker ps --no-trunc** command on the container instance.

```
curl http://localhost:51678/v1/tasks?dockerid=79c796ed2a7f
```

Output:

```
{
    "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/e01d58a8-151b-40e8-
bc01-22647b9ecfec",
    "Containers": [
        {
            "DockerId": "79c796ed2a7f864f485c76f83f3165488097279d296a7c05bd5201a1c69b2920",
            "DockerName": "ecs-nginx-efs-2-nginx-9ac0808dd0afa495f001",
            "Name": "nginx"
        }
```

```
    ],
    "DesiredStatus": "RUNNING",
    "Family": "nginx-efs",
    "KnownStatus": "RUNNING",
    "Version": "2"
}
```

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. 311)
- Windows Container Instance Configuration (p. 314)

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=169.254.169.254,169.254.170.2,/var/run/docker.sock`

Set this value to `169.254.169.254,169.254.170.2,/var/run/docker.sock` to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

`/etc/systemd/system/docker.service.d/http-proxy.conf` (Amazon Linux 2 only)

`Environment="HTTP_PROXY=http://10.0.0.131:3128"`

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Docker daemon to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

`Environment="NO_PROXY=169.254.169.254"`

Set this value to `169.254.169.254` to filter EC2 instance metadata from the proxy.

`/etc/sysconfig/docker` (Amazon Linux AMI and Amazon Linux 2 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=169.254.169.254,169.254.170.2`

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. 219), and in Step 6.g (p. 220). 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.

```
#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"
elif 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."
fi

# 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:$PROXY_PORT/"
Environment="NO_PROXY=169.254.169.254,169.254.170.2"
EOF
    systemctl daemon-reload
    if [ "$(systemctl is-active docker)" == "active" ]
    then
        systemctl restart docker
    fi
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_docker_http_proxy
fi
# Amazon Linux AMI
if [ $OS == "ALAMI" ] && [ ! -f /var/lib/cloud/instance/sem/config_docker_http_proxy ];
 then
    echo "export HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/" >> /etc/sysconfig/docker
    echo "export HTTPS_PROXY=https://$PROXY_HOST:$PROXY_PORT/" >> /etc/sysconfig/docker
    echo "export NO_PROXY=169.254.169.254,169.254.170.2" >> /etc/sysconfig/docker
    echo "$$: $(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
if [ $OS == "ALAMI" ] && [ ! -f /var/lib/cloud/instance/sem/config_ecs-init_http_proxy ];
 then
    cat <<EOF > /etc/init/ecs.override
env HTTP_PROXY=$PROXY_HOST:$PROXY_PORT
env 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-
init_http_proxy
fi
```

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).

```
[Environment]::SetEnvironmentVariable("HTTP_PROXY",
"http://proxy.mydomain:port", "Machine")
```

Set `HTTP_PROXY` to the hostname (or IP address) and port number of an HTTP proxy to use for the 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 Launching an Amazon ECS Container Instance (p. 219). Just copy and paste the PowerShell script below into the **User data** field (be sure to substitute the red example values with your own proxy and cluster information).

Note
The `-EnableTaskIAMRole` option is required to enable IAM roles for tasks. For more information, see Windows IAM roles for tasks (p. 721).

```
<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>
```

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. 339).
- `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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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. 339).

  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 Amazon ECS services (p. 338).

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. 316) and RunTask in the *Amazon Elastic Container Service API Reference*.

Running Tasks on a `cron`-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. 330).

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. 320).

Contents

- Running Tasks (p. 316)
- Amazon ECS Task Placement (p. 320)
- Scheduled Tasks (cron) (p. 330)
- Task Lifecycle (p. 332)
- Task Retirement (p. 334)
- Fargate Task Recycling (p. 336)
- Creating a Scheduled Task Using the AWS CLI (p. 336)

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 Amazon ECS services (p. 338).

Contents

- Running a Task Using the Fargate Launch Type (p. 316)
- Running a Task Using the EC2 Launch Type (p. 318)

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. 119).
   b. For **Platform version**, choose **LATEST**. For more information about platform versions, see AWS Fargate Platform Versions (p. 33).
   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, environment variable, task IAM role, or task execution role overrides to one or more containers in your task definition, choose **Advanced Options** and complete the following steps:

     Note
     If you will be using the parameter values from your task definition there is no need to specify overrides. These fields are only used to override the values specified in the task definition.

     i. For **Task Role Override**, choose an IAM role for this task to override the task IAM role specified in the task definition. For more information, see IAM Roles for Tasks (p. 489).

     Only roles with the `ecs-tasks.amazonaws.com` 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. 491).

     ii. For **Task Execution Role Override**, choose a task execution role to override the task execution role specified in the task definition. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

     iii. For **Container Overrides**, choose a container to which to send a command or environment variable override.

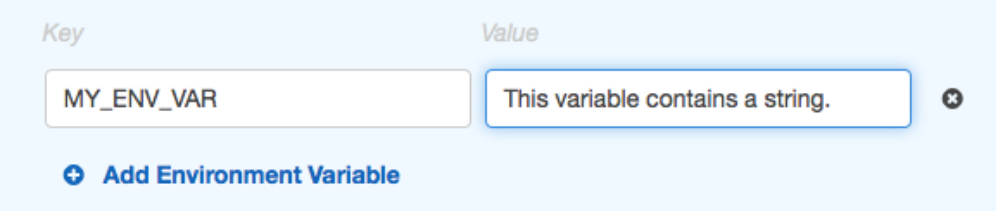
     - **For a command override:** For **Command override**, type the command override to send. If your container definition does not specify an `ENTRYPOINT`, the format should be a comma-separated list of non-quoted strings. For example:

```
/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:

```
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).



This environment variable override is sent to the container as:

```
MY_ENV_VAR="This variable contains a string."
```

7. 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.

8. 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.
9. 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. 690) in the troubleshooting section.

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. 119).
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:

   - **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. 320) for examples.

   For more information, see Amazon ECS Task Placement (p. 320).
8. (Optional) To send command, environment variable, task IAM role, or task execution role overrides to one or more containers in your task definition, choose **Advanced Options** and complete the following steps:

   Note
   If you will be using the parameter values from your task definition there is no need to specify overrides. These fields are only used to override the values specified in the task definition.

   a. For **Task Role Override**, choose an IAM role for this task to override the task IAM role specified in the task definition. For more information, see IAM Roles for Tasks (p. 489).

      Only roles with the `ecs-tasks.amazonaws.com` 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. 491).

   b. For **Task Execution Role Override**, choose a task execution role to override the task execution role specified in the task definition. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

   c. For **Container Overrides**, choose a container to which to send a command or environment variable override.

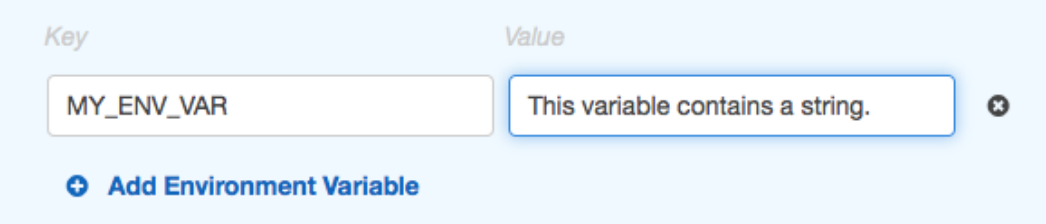
- **For a command override:** For **Command override**, type the command override to send. If your container definition does not specify an `ENTRYPOINT`, the format should be a comma-separated list of non-quoted strings. For example:

```
/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:

```
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).



  This environment variable override is sent to the container as:

```
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. 690) 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 use a task placement strategy and a task placement constraint to distribute tasks across Availability Zones and bin pack tasks based on memory within each Availability Zone, but only for G2 instances.

When Amazon ECS places tasks, it uses the following process to select container instances:

1. Identify the 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. 321)
- Amazon ECS Task Placement Constraints (p. 323)
- Cluster Query Language (p. 327)

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. The task placement strategies can be updated for existing services as well. For more information, see Amazon ECS Task Placement (p. 320).

Strategy Types

Amazon ECS supports the following task placement strategies:

`binpack`

Tasks are placed on container instances so as to leave the least amount of unused CPU or memory. This strategy minimizes the number of container instances in use.

When this strategy is used and a scale-in action is taken, Amazon ECS will terminate tasks based on the amount of resources that will be left on the container instance after the task is terminated. The contianer instance that will have the most available resources left after task termination will have that task terminated.

`random`

Tasks are placed randomly.

`spread`

Tasks are placed evenly based on the specified value. Accepted 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`. Service tasks are spread based on the tasks from that service. Standalone tasks are spread based on the tasks from the same task group.

When this strategy is used and a scale-in action is taken, Amazon ECS will terminate tasks based on age, with the newest tasks being terminated first. If age cannot be considered because multiple tasks have the same age, then the tasks will be chosen at random.

Example Strategies

You can specify task placement strategies with the following actions: CreateService, UpdateService, 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",
        "type": "spread"
    }
]
```

The following strategy distributes tasks evenly across Availability Zones and then bin packs tasks based on memory within each Availability Zone.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

Amazon ECS Task Placement Constraints

A *task placement constraint* is a rule that is considered during task placement. Task placement constraints can be specified when either running a task or creating a new service. The task placement constraints can be updated for existing services as well. For more information, see Amazon ECS Task Placement (p. 320).

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. 327).

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-1234abcd`.

`ecs.availability-zone`

The Availability Zone for the instance. An example value for this attribute is `us-east-1a`.

`ecs.instance-type`

The instance type for the instance. An example value for this attribute is `g2.2xlarge`.

`ecs.os-type`

The operating system for the instance. The possible values for this attribute are `linux` and `windows`.

`ecs.cpu-architecture`

The CPU architecture for the instance. The possible values for this attribute are `x86_64` and `arm64`.

`ecs.vpc-id`

The VPC the instance was launched into. An example value for this attribute is `vpc-1234abcd`.

`ecs.subnet-id`

The subnet the instance is using. An example value for this attribute is `subnet-1234abcd`.

Optional Attributes

Amazon ECS may add the following attribute to your container instances.

`ecs.awsvpc-trunk-id`

If this attribute exists, the instance has a trunk network interface. For more information, see Elastic Network Interface Trunking (p. 230).

`ecs.outpost-arn`

If this attribute exists, it contains the Amazon Resource Name (ARN) of the Outpost. For more information, see *Amazon Elastic Container Service on AWS Outposts* (p. 619).

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. 278).

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.
4. Choose **Actions**, **View/Edit Attributes**.

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-constainer-instances command. For more information about the filter syntax, see Cluster Query Language (p. 327).

Example: Built-in Attribute

The following example uses built-in attributes to list the g2.2xlarge instances.

```
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".

```
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".

```
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`.

```
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.

```
aws ecs list-container-instances --filter "attribute:ecs.instance-type =~ t2.* and
 attribute:ecs.availability-zone == us-east-1a"
```

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, UpdateService, 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, UpdateService, RegisterTaskDefinition, and RunTask.

```
"placementConstraints": [
    {
        "expression": "task:group == databases",
        "type": "memberOf"
    }
]
```

The `distinctInstance` constraint places each task in the group on a different instance. It can be specified with the following actions: CreateService, UpdateService, and RunTask

```
"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. 323).

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. 316) and Creating a service (p. 352). You can also apply a group filter when listing container instances. For more information, see Filtering by Attribute (p. 325).

Expression Syntax

Expressions have the following syntax:

```
subject operator [argument]
```

Subject

The attribute or field to be evaluated.

`agentConnected`

Select container instances by their Amazon ECS container agent connection status. You can use this filter to search for instances with container agents that are disconnected.

Valid operators: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`agentVersion`

Select container instances by their Amazon ECS container agent version. You can use this filter to find instances that are running outdated versions of the Amazon ECS container agent.

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`attribute:`*`attribute-name`*

Select container instances by attribute. For more information, see Attributes (p. 323).

`ec2InstanceId`

Select container instances by their Amazon EC2 instance ID.

Valid operators: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`registeredAt`

Select container instances by their container instance registration date. You can use this filter to find newly registered instances or instances that are very old.

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

Valid date formats: 2018-06-18T22:28:28+00:00, 2018-06-18T22:28:28Z, 2018-06-18T22:28:28, 2018-06-18

`runningTasksCount`

Select container instances by number of running tasks. You can use this filter to find instances that are empty or near empty (few tasks running on them).

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`task:group`

Select container instances by task group. For more information, see Task Groups (p. 326).

Operator

The comparison operator. The following operators are supported.

Operator	Description
==, equals	String equality
!=, not_equals	String inequality
>, greater_than	Greater than
>=, greater_than_equal	Greater than or equal to
<, less_than	Less than
<=, less_than_equal	Less than or equal to
exists	Subject exists
!exists, not_exists	Subject does not exist
in	Value in argument list
!in, not_in	Value not in argument list
=~, matches	Pattern match
!~, not_matches	Pattern mismatch

Note
A single expression can't contain parentheses. However, parentheses can be used to specify precedence in compound expressions.

Argument

For many operators, the argument is a literal value.

The `in` and `not_in` operators expect an argument list as the argument. You specify an argument list as follows:

```
[argument1, argument2, ..., argumentN]
```

The matches and not_matches operators expect an argument that conforms to the Java regular expression syntax. For more information, see java.util.regex.Pattern.

Compound Expressions

You can combine expressions using the following Boolean operators:

- &&, and
- ||, or
- !, not

You can specify precedence using parentheses:

```
(expression1 or expression2) and expression3
```

Example Expressions

The following are example expressions.

Example: String Equality

The following expression selects instances with the specified instance type.

```
attribute:ecs.instance-type == t2.small
```

Example: Argument List

The following expression selects instances in the us-east-1a or us-east-1b Availability Zone.

```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

Example: Compound Expression

The following expression selects G2 instances that are not in the us-east-1d Availability Zone.

```
attribute:ecs.instance-type =~ g2.* and attribute:ecs.availability-zone != us-east-1d
```

Example: Task Affinity

The following expression selects instances that are hosting tasks in the `service:production` group.

```
task:group == service:production
```

Example: Task Anti-Affinity

The following expression selects instances that are not hosting tasks in the database group.

```
not(task:group == database)
```

Example: Running task count

The following expression selects instances that are only running one task.

```
runningTasksCount == 1
```

Example: Amazon ECS container agent version

The following expression selects instances that are running a container agent version below 1.14.5.

```
agentVersion < 1.14.5
```

Example: Instance registration time

The following expression selects instances that were registered before February 13, 2018.

```
registeredAt < 2018-02-13
```

Example: Amazon EC2 instance ID

The following expression selects instances with the following Amazon EC2 instance IDs.

```
ec2InstanceId in ['i-abcd1234', 'i-wxyx7890']
```

Scheduled Tasks (cron)

Amazon ECS supports the ability to schedule tasks on either a `cron`-like schedule or in a response to CloudWatch Events. This is supported for Amazon ECS tasks using both the Fargate and EC2 launch types.

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*.

You can also now set your Fargate tasks as a task target in CloudWatch Events, allowing you to launch tasks in response to changes that happen. Additionally, you can modify the network configuration when using the `awsvpc` network mode via the CloudWatch Events console and AWS CLI, giving Fargate tasks triggered by CloudWatch Events the same networking properties as Amazon EC2 instances. For more information, see Tutorial: Run an Amazon ECS Task When a File is Uploaded to an Amazon S3 Bucket in the *Amazon CloudWatch Events User Guide*.

Note
This feature is not yet available for Fargate tasks in the following Regions:

Region Name	Region
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
Middle East (Bahrain)	me-south-1

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. If you do not have any clusters, see Creating a Cluster (p. 38) for steps on creating a new cluster.
3. On the **Cluster: *`cluster-name`*** page, choose **Scheduled Tasks**, **Create**.
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.
   a. For **Target id**, enter a unique identifier for your target. Up to 64 letters, numbers, periods, hyphens, and underscores are allowed.
   b. For **Launch type**, choose the launch type for the tasks in your service. For more information, see Amazon ECS Launch Types (p. 119).
   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 (p. 33).

      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 (p. 489). 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. 491). You must add `iam:PassRole` permissions for any task role and task role overrides to the CloudWatch IAM role. For more information, see Amazon ECS CloudWatch Events IAM Role (p. 496).

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 (p. 141)..

   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 Amazon ECS CloudWatch Events IAM Role (p. 496).

   Important
   If your scheduled tasks require the use of the task execution role, a task role, or if they use a task role override, then you must add `iam:PassRole` permissions for your task execution role, task role, or task role override to the CloudWatch IAM role. For more information, see Amazon ECS CloudWatch Events IAM Role (p. 496).

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.
3. On the **Cluster: *`cluster-name`*** page, choose **Scheduled Tasks**.
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**.

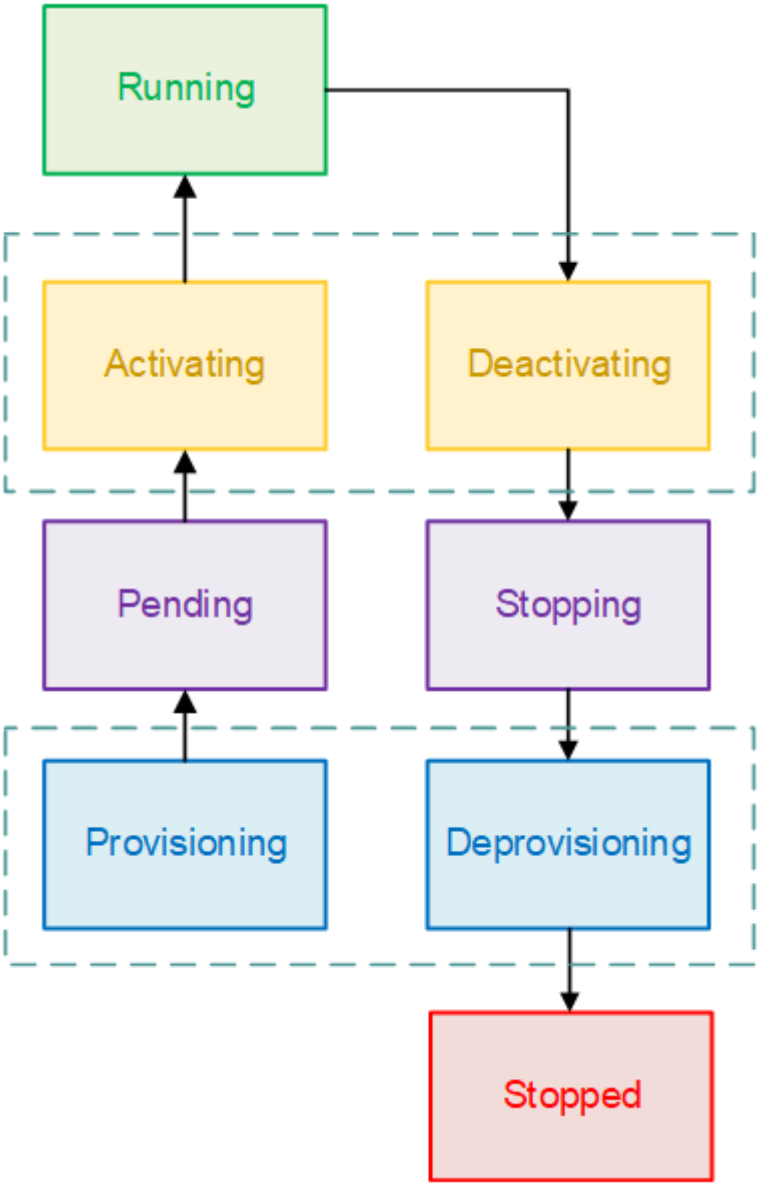
Task Lifecycle

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. For tasks that are part of a service that is configured to use multiple Elastic Load Balancing target groups, the target group registration occurs during this state.

RUNNING

The task is successfully running.

DEACTIVATING

Amazon ECS has to perform additional steps before the task is stopped. For example, for tasks that are part of a service that is configured to use multiple Elastic Load Balancing target groups, the target group deregistration occurs during this state.

STOPPING

This is a transition state where Amazon ECS is waiting on the container agent to take further action.

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

Amazon ECS task retirement affects tasks of both Fargate and EC2 launch types and you will be notified by email of the pending retirement.

A task can be scheduled for retirement in the following scenarios:

- AWS detects the irreparable failure of the underlying hardware hosting the task.
- Your task uses the Fargate launch type and is running on a platform version that has a security vulnerability that requires you to replace the tasks by launching new tasks using a patched platform version.

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.

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 launches 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.

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. 363).

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.
5. 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. 33).
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.

   ```
   aws ecs list-services --cluster cluster_name --region region
   ```

   Output:

   ```
   {
       "serviceArns": [
           "arn:aws:ecs:region:aws_account_id:service/MyService"
       ]
   }
   ```
2. Update your service, forcing a new deployment that deploys new tasks.

   ```
   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. 316).

Fargate Task Recycling

Amazon ECS task recycling only affects tasks using the Fargate and no notification is sent prior to the recycling event.

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. 33).

  Note
  Fargate tasks using platform versions prior to 1.3.0 are not affected.
- 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. 334).
- 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 recommending 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}"
```

Amazon ECS services

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

In addition to maintaining the desired number 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. 338)
- Additional service concepts (p. 340)
- Service Definition Parameters (p. 340)
- Creating a service (p. 352)
- Updating a Service (p. 363)
- Deleting a Service (p. 366)
- Amazon ECS Deployment Types (p. 366)
- Service Load Balancing (p. 376)
- Service Auto Scaling (p. 395)
- Service Discovery (p. 403)
- Service Throttle Logic (p. 406)

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 (also referred to as the *service type*) that you specified when creating the service.

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. 406) and Service Event Messages (p. 694).

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. 339).
- `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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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. 339).

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. The service scheduler also evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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 a `DRAINING` status. If a container instance transitions to `DRAINING`, the daemon tasks on it are stopped. The service scheduler also monitors when new container instances are added to your cluster and adds the daemon tasks to them.

If a `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.

Tasks using the Fargate launch type or the `CODE_DEPLOY` or `EXTERNAL` deployment controller types don't support the daemon scheduling strategy.

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 in your cluster.

When using the Fargate launch type with tasks, when the service scheduler launches new tasks or stops running tasks, it attempts to maintain balance across Availability Zones.

When using the EC2 launch type with tasks, you can optionally specify task placement strategies and constraints to customize task placement decisions. If no task placement strategies or constraints are specified, then by default the service scheduler will spread the tasks across Availability Zones. 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 stops running tasks that use the EC2 launch type, it attempts to maintain balance across the Availability Zones in your cluster. The scheduler uses the following logic:

- If a placement strategy is defined, use that strategy to select which tasks to terminate. For example, if a service has an Availability Zone spread strategy defined, 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. 376).
- You can optionally specify a deployment configuration for your service. A deployment is triggered by updated the task definition or desired count of a service. During a deployment, 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. 340).
- 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. 403).
- When you delete a service, if there are still running tasks that require cleanup, the service status moves from `ACTIVE` to `DRAINING`, and the service is no longer visible in the console or in the ListServices API operation. After all tasks have transitioned to either `STOPPING` or `STOPPED` status, the service status moves from `DRAINING` to `INACTIVE`. Services in the `DRAINING` or `INACTIVE` status can still be viewed with the DescribeServices API operation. However, in the future, `INACTIVE` services may be cleaned up and purged from Amazon ECS record keeping, and DescribeServices calls on those services return a `ServiceNotFoundException` error.

Service Definition Parameters

A service definition defines how to run your Amazon ECS service. The following parameters can be specified in a service definition.

Launch Type

`launchType`

Type: String

Valid values: `EC2` | `FARGATE`

Required: No

The launch type on which to run your service. If a launch type is not specified, `EC2` is used by default. For more information, see Amazon ECS Launch Types (p. 119).

If a `launchType` is specified, the `capacityProviderStrategy` parameter must be omitted.

Capacity Provider Strategy

`capacityProviderStrategy`

Type: Array of objects

Required: No

The capacity provider strategy to use for the service.

A capacity provider strategy consists of one or more capacity providers along with the `base` and `weight` to assign to them. A capacity provider must be associated with the cluster to be used in a capacity provider strategy. The PutClusterCapacityProviders API is used to associate a capacity provider with a cluster. Only capacity providers with an `ACTIVE` or `UPDATING` status can be used.

If a `capacityProviderStrategy` is specified, the `launchType` parameter must be omitted. If no `capacityProviderStrategy` or `launchType` is specified, the `defaultCapacityProviderStrategy` for the cluster is used.

If specifying a capacity provider that uses an Auto Scaling group, the capacity provider must already be created. New capacity providers can be created with the CreateCapacityProvider API operation.

To use a AWS Fargate capacity provider, specify either the `FARGATE` or `FARGATE_SPOT` capacity providers. The AWS Fargate capacity providers are available to all accounts and only need to be associated with a cluster to be used.

The PutClusterCapacityProviders API operation is used to update the list of available capacity providers for a cluster after the cluster is created.

`capacityProvider`

Type: String

Required: Yes

The short name or full ARN of the capacity provider.

`weight`

Type: Integer

Valid range: Integers between 0 and 1,000.

Required: No

The weight value designates the relative percentage of the total number of tasks launched that should use the specified capacity provider.

For example, if you have a strategy that contains two capacity providers and both have a weight of 1, then when the base is satisfied, the tasks will be split evenly across the two capacity providers. Using that same logic, if you specify a weight of 1 for *capacityProviderA* and a weight of 4 for *capacityProviderB*, then for every one task that is run using *capacityProviderA*, four tasks would use *capacityProviderB*.

`base`

Type: Integer

Valid range: Integers between 0 and 100,000.

Required: No

The base value designates how many tasks, at a minimum, to run on the specified capacity provider. Only one capacity provider in a capacity provider strategy can have a base defined.

Task Definition

`taskDefinition`

Type: String

Required: No

The `family` and `revision` (`family:revision`) or full Amazon Resource Name (ARN) of the task definition to run in your service. If a `revision` is not specified, the latest `ACTIVE` revision of the specified family is used.

A task definition must be specified when using the rolling update (`ECS`) deployment controller.

Platform Version

`platformVersion`

Type: String

Required: No

The platform version on which your tasks in the service are running. A platform version is only specified for tasks using the Fargate launch type. If one is not specified, the latest version (`LATEST`) is used by default.

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. 33).

Note
Platform versions are not specified for tasks using the EC2 launch type.

Cluster

`cluster`

Type: String

Required: No

The short name or full Amazon Resource Name (ARN) of the cluster on which to run your service. If you do not specify a cluster, the `default` cluster is assumed.

Service Name

`serviceName`

Type: String

Required: Yes

The name of your service. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a Region or across multiple Regions.

Scheduling Strategy

`schedulingStrategy`

Type: String

Valid values: `REPLICA` | `DAEMON`

Required: No

The scheduling strategy to use. If no scheduling strategy is specified, the `REPLICA` strategy is used. For more information, see Service scheduler concepts (p. 338).

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. 339).
- `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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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. 339).

  Note
  Fargate tasks do not support the `DAEMON` scheduling strategy.

Desired Count

`desiredCount`

Type: Integer

Required: No

The number of instantiations of the specified task definition to place and keep running on your cluster.

This parameter is required if the `REPLICA` scheduling strategy is used. If the service uses the `DAEMON` scheduling strategy, this parameter is optional.

Deployment Configuration

`deploymentConfiguration`

Type: Object

Required: No

Optional deployment parameters that control how many tasks run during the deployment and the ordering of stopping and starting tasks.

`maximumPercent`

Type: Integer

Required: No

If a service is using the rolling update (`ECS`) deployment type, the `maximumPercent` parameter represents an upper limit on the number of your service's tasks that are allowed in the `RUNNING` 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 **maximum percent** 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`

Type: Integer

Required: No

If a service is using the rolling update (`ECS`) deployment type, the `minimumHealthyPercent` represents a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment, 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.

For services that *do not* use a load balancer, the following should be noted:

- A service is considered healthy if all essential containers within the tasks in the service pass their health checks.
- If a task has no essential containers with a health check defined, the service scheduler will wait for 40 seconds after a task reaches a `RUNNING` state before the task is counted towards the minimum healthy percent total.
- If a task has one or more essential containers with a health check defined, the service scheduler will wait for the task to reach a healthy status before counting it towards the minimum healthy percent total. A task is considered healthy when all essential containers within the task have passed their health checks. The amount of time the service scheduler can wait for is determined by the container health check settings. For more information, see Health Check (p. 90).

For services are that *do* use a load balancer, the following should be noted:

- If a task has no essential containers with a health check defined, the service scheduler will wait for the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.
- If a task has an essential container with a health check defined, the service scheduler will wait for both the task to reach a healthy status and the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.

The default value for a replica service for `minimumHealthyPercent` is 100%. The default `minimumHealthyPercent` value for a service using the `DAEMON` service schedule is 0% for the AWS CLI, the AWS SDKs, and the APIs and 50% for the AWS Management Console.

The minimum number of healthy tasks during a deployment is the `desiredCount` multiplied by the `minimumHealthyPercent`/100, rounded up to the nearest integer value.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and 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.

Deployment Controller

`deploymentController`

Type: Object

Required: No

The deployment controller to use for the service. If no deploymenet controller is specified, the `ECS` controller is used. For more information, see Amazon ECS Deployment Types (p. 366).

`type`

Type: String

Valid values: `ECS` | `CODE_DEPLOY` | `EXTERNAL`

Required: yes

The deployment controller type to use. There are three deployment controller types available:

`ECS`

The rolling update (`ECS`) deployment type involves replacing the current running version of the container with the latest version. The number of containers Amazon ECS adds or removes from the service during a rolling update is controlled by adjusting the minimum and maximum number of healthy tasks allowed during a service deployment, as specified in the deploymentConfiguration.

`CODE_DEPLOY`

The blue/green (`CODE_DEPLOY`) deployment type uses the blue/green deployment model powered by CodeDeploy, which allows you to verify a new deployment of a service before sending production traffic to it.

`EXTERNAL`

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.

Task Placement

`placementConstraints`

Type: Array of objects

Required: No

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task (this limit includes constraints in the task definition and those specified at run time). If you are using the Fargate launch type, task placement constraints are not supported.

`type`

Type: String

Required: No

The type of constraint. Use `distinctInstance` to ensure that each task in a particular group is running on a different container instance. Use `memberOf` to restrict the selection to a group of valid candidates. The value `distinctInstance` is not supported in task definitions.

`expression`

Type: String

Required: No

A cluster query language expression to apply to the constraint. Note you cannot specify an expression if the constraint type is `distinctInstance`. For more information, see Cluster Query Language (p. 327).

`placementStrategy`

Type: Array of objects

Required: No

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

`type`

Type: String

Valid values: `random` | `spread` | `binpack`

Required: No

The type of placement strategy. The `random` placement strategy randomly places tasks on available candidates. The `spread` placement strategy spreads placement across available candidates evenly based on the `field` parameter. The `binpack` strategy places tasks on available candidates that have the least available amount of the resource that 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`

Type: String

Required: No

The field to apply the placement strategy against. For the `spread` placement strategy, valid values are `instanceId` (or `host`, which has the same effect), or any platform or custom

attribute that 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.

Tags

`tags`

Type: Array of objects

Required: No

The metadata that you apply to the service to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see Tagging Your Amazon ECS Resources (p. 407).

`key`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

`value`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

`enableECSManagedTags`

Type: Boolean

Valid values: `true` | `false`

Required: No

Specifies whether to enable Amazon ECS managed tags for the tasks in the service. If no value is specified, the default value is `false`. For more information, see Tagging Your Resources for Billing (p. 409).

`propagateTags`

Type: String

Valid values: `TASK_DEFINITION` | `SERVICE`

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation, use the `TagResource` API action.

Network Configuration

`networkConfiguration`

Type: Object

Required: No

The network configuration for the service. This parameter is required for task definitions that use the `awsvpc` network mode to receive their own Elastic Network Interface, and it 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. 141)..

`awsvpcConfiguration`

Type: Object

Required: No

An object representing the subnets and security groups for a task or service.

`subnets`

Type: Array of strings

Required: Yes

The subnets associated with the task or service. There is a limit of 16 subnets that can be specified per `awsvpcConfiguration`.

`securityGroups`

Type: Array of strings

Required: No

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. There is a limit of 5 security groups that can be specified per `awsvpcConfiguration`.

`assignPublicIP`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether the task's elastic network interface receives a public IP address. If no value is specified, the default value of `DISABLED` is used.

`healthCheckGracePeriodSeconds`

Type: Integer

Required: No

The period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks, container health checks, and Route 53 health checks after a task enters a `RUNNING` state. This is only valid if your service is configured to use a load balancer. If your service has a load balancer defined and you do not specify a health check grace period value, the default value of `0` is used.

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.

`loadBalancers`

Type: Array of objects

Required: No

A load balancer object representing the load balancers to use with your service. For services that use an Application Load Balancer or Network Load Balancer, there is a limit of five target groups you can attach to a 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`

Type: String

Required: No

The full ARN of the Elastic Load Balancing target group associated with a service.

A target group ARN is only specified when using an Application Load Balancer or Network Load Balancer. If you are using a Classic Load Balancer the target group ARN should be omitted.

`loadBalancerName`

Type: String

Required: No

The name of the load balancer to associate with the service.

A load balancer name is only specified when using a Classic Load Balancer. If you are using an Application Load Balancer or a Network Load Balancer the load balancer name parameter should be omitted.

`containerName`

Type: String

Required: No

The name of the container (as it appears in a container definition) to associate with the load balancer.

`containerPort`

Type: Integer

Required: No

The port on the container to associate with the load balancer. This port must correspond to a `containerPort` in the task definition used by tasks in the service. For tasks that use the EC2 launch type, the container instance must allow ingress traffic on the `hostPort` of the port mapping.

`role`

Type: String

Required: No

The short name or full ARN of the IAM role that allows Amazon ECS to make calls to your load balancer on your behalf. This parameter is only permitted if you are using a load balancer with 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 Service-Linked Role for Amazon ECS (p. 473).

`serviceRegistries`

Type: Array of objects

Required: No

The details of the service discovery configuration for your service. For more information, see Service Discovery (p. 403).

`registryArn`

Type: String

Required: No

The ARN of the service registry. The currently supported service registry is AWS Cloud Map. For more information, see Working with Services in the *AWS Cloud Map Developer Guide*.

`port`

Type: Integer

Required: No

The port value used if your service discovery service specified an SRV record. This field is required if both the `awsvpc` network mode and SRV records are used.

`containerName`

Type: String

Required: No

The container name value, 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`

Type: Integer

Required: No

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. If the task definition your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

Client Token

`clientToken`

Type: String

Required: No

Unique, case-sensitive identifier you provide to ensure the idempotency of the request. Up to 32 ASCII characters are allowed.

Service Definition Template

The following shows the JSON representation of an Amazon ECS service definition.

```
{
    "cluster": "",
    "serviceName": "",
    "taskDefinition": "",
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "loadBalancerName": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "desiredCount": 0,
    "clientToken": "",
    "launchType": "FARGATE",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
```

```
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "role": "",
    "deploymentConfiguration": {
        "maximumPercent": 0,
        "minimumHealthyPercent": 0
    },
    "placementConstraints": [
        {
            "type": "distinctInstance",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "spread",
            "field": ""
        }
    ],
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "ENABLED"
        }
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "SERVICE"
}
```

You can create this service definition template using the following AWS CLI command.

```
aws ecs create-service --generate-cli-skeleton
```

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. 376). 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. 353)
- Step 2: Configure a Network (p. 355)
- Step 3: Configuring Your Service to Use a Load Balancer (p. 356)
- Step 4: Configuring Your Service to Use Service Discovery (p. 360)
- Step 5: Configuring Your Service to Use Service Auto Scaling (p. 361)
- Step 6: Review and Create Your Service (p. 363)

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. 340).

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.

> **Note**
> If your cluster is configured with a default capacity provider strategy, you will only be able to create a service using the default capacity provider strategy when using the console. Likewise, if no default capacity provider is defined, you will only be able to use a launch type when creating a service using the console. It is not currently possible to have a mixed strategy using both capacity providers and launch types in the console.

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:

   - **Capacity provider strategy**: Choose whether your service should use the default capacity provider strategy defined for the cluster or a custom capacity provider strategy. A capacity provider must already be associated with the cluster in order to be used in a custom capacity provider strategy. For more information, see Amazon ECS Cluster Capacity Providers (p. 40).
   - **Launch type**: Choose whether your service should run tasks on Fargate infrastructure, or Amazon EC2 container instances that you maintain. This option is available if your cluster has no default capacity provider defined. For more information, see Amazon ECS Launch Types (p. 119).
   - **Platform version**: If you chose the Fargate launch type, then select the platform version to use.

     > **Note**
     > When the **LATEST** platform version is selected, the `1.3.0` platform version is used. To use platform version `1.4.0`, you must select the **1.4.0** option.

- **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. 338).
- **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. 87).
- If you are using the **Rolling update** deployment type, fill out the following deployment configuration parameters. For more information on how these parameters are used, see Deployment Configuration (p. 343).
  - **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).
  - **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).

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. 366).
   - If you selected the blue/green deployment type, complete the following steps:
     - For **Deployment configuration** choose the deployment configuration to use for the service. This determines how traffic is shifted when your task set is updated. For more information, see Blue/Green Deployment with CodeDeploy (p. 367)
     - For **Service role for CodeDeploy** choose the IAM service role for AWS CodeDeploy. For more information, see Amazon ECS CodeDeploy IAM Role (p. 493)
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. 320) for examples.

   For more information, see Amazon ECS Task Placement (p. 320).
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. 355).

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 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. 356).

The `awsvpc` network mode does not provide task ENIs 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 internet 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.

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. 353).
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.
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, in order for the task to pull the container image it must either use a public subnet and be assigned a public IP address or a private subnet that has a route to the internet or a NAT gateway that can route requests to the internet.
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. 356).
   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: Configuring Your Service to Use Service Auto Scaling (p. 361).

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: Configuring Your Service to Use Service Discovery (p. 360).

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. 382).

> **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. 356)
- Configuring a Load Balancer for the Blue/Green Deployment Type (p. 358)

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. 353).
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. 353).
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 Service Scheduler IAM Role (p. 479).

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. 357). If you've chosen a Network Load Balancer, follow the steps in To configure a Network Load Balancer (p. 358). If you've chosen a Classic Load Balancer, follow the steps in To configure a Classic Load Balancer (p. 358).

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. 383) (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. 383) (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 a Network Load Balancer (p. 387) (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 a Network Load Balancer (p. 387) (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 **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. 389) (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. 353).
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. 357). If you've chosen a Network Load Balancer, follow the steps in To configure a Network Load Balancer (p. 358).

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. 383) (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. 383) (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: Configuring Your Service to Use Service Discovery (p. 360).

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. 383) (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. 383) (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: Configuring Your Service to Use Service Discovery (p. 360).

Step 4: 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. 403).

If you are not configuring your service to use a service discovery, you can move on to the next section, Step 5: Configuring Your Service to Use Service Auto Scaling (p. 361).

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. 353).
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 Supported DNS Record Types in the *Amazon Route 53 Developer Guide*.

- If the task definition that your service task specifies uses the `bridge` or `host` network mode, only type SRV records are supported. Choose a container name and port combination to associate with the record.
- If the task definition that your service task specifies uses the `awsvpc` network mode, select either the A or SRV record type. If 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: Configuring Your Service to Use Service Auto Scaling (p. 361).

Step 5: 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 of tasks in your Amazon ECS service 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. 396) (Recommended)—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. 401)—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. 395).

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. 353).
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 the `ecsAutoscaleRole`. If this role does not exist, choose **Create new role** to have the console create it for you.
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.

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. The following metrics are available:
   - **ECSServiceAverageCPUUtilization**—Average CPU utilization of the service.
   - **ECSServiceAverageMemoryUtilization**—Average memory utilization of the service.
   - **ALBRequestCountPerTarget**—Number of requests completed per target in an Application Load Balancer target group.
4. For **Target value**, enter the metric value that the policy should maintain. For example, use a target value of `1000` for `ALBRequestCountPerTarget`, or a target value of `75`(%) for `ECSServiceAverageCPUUtilization`.
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. While the scale-out cooldown period is in effect, the capacity that has been added by the previous scale-out activity that initiated the cooldown is calculated as part of the desired capacity for the next scale out. The intention is to continuously (but not excessively) scale out.
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. The scale-in cooldown period is used to block subsequent scale-in requests until it has expired. The intention is to scale in conservatively to protect your application's availability. However, if another alarm triggers a scale out activity during the cooldown period after a scale-in, Service Auto Scaling scales out your scalable target immediately.
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 Auto Scaling (p. 395).
   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. 363) 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. 362) through Step 8 (p. 363) 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. 363).

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 an existing service to change some of the service configuration parameters, such as 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 want to use an updated container image for your tasks, you can create a new task definition revision with that image and deploy it to your service by using the **force new deployment** option in the console.

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 `SIGKILL` is sent and the containers are forcibly stopped. If the container handles the `SIGTERM` signal gracefully and exits within 30 seconds from receiving it, no `SIGKILL` signal is sent. The service scheduler starts and stops tasks as defined by your minimum healthy percent and maximum percent settings.

Important
If you are changing the ports used by containers in a task definition, you may need to update the security groups for the container instances 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.
5. 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. Change the task definition, capacity provider strategy, platform version, deployment configuration, or number of desired tasks (or any combination of these). To have your service start a new deployment, which will stop and relaunch all tasks using the new configuration, select **Force new deployment**. Choose **Next step** when finished changing the service configuration.
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*.
9. Choose **Next step**.
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**.
11. (Optional) You can use Service Auto Scaling to scale your service up and down automatically in response to CloudWatch alarms.

    a. Under **Optional configurations**, choose **Configure Service Auto Scaling**.

    b. Proceed to Step 5: Configuring Your Service to Use Service Auto Scaling (p. 361).

    c. Complete the steps in that section and then return.
12. Choose **Update Service** to finish and update your service.

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.
4. On the **Cluster :** ***name*** page, choose **Services**.
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.

```
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.

```
aws servicediscovery delete-service --id <service_discovery_service_id> --
region <region_name>
```

4. Retrieve the ID of the service discovery namespace to delete.

```
aws servicediscovery list-namespaces --region <region_name>
```

5. Using the service discovery namespace ID from the previous output, delete the namespace.

```
aws servicediscovery delete-namespace --id <service_discovery_namespace_id> --
region <region_name>
```

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. 367)
- Blue/Green Deployment with CodeDeploy (p. 367)
- External Deployment (p. 371)

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. 352).

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*.

There are three ways traffic can shift during a blue/green deployment:

- **Canary** — Traffic is shifted in two increments. You can choose from predefined canary options that specify the percentage of traffic shifted to your updated task set in the first increment and the interval, in minutes, before the remaining traffic is shifted in the second increment.
- **Linear** — Traffic is shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment.
- **All-at-once** — All traffic is shifted from the original task set to the updated task set all at once.

The following are components of CodeDeploy that Amazon ECS uses when a service uses the blue/green deployment type:

CodeDeploy application

A collection of CodeDeploy resources. This consists of one or more deployment groups.

CodeDeploy deployment group

The deployment settings. This consists of the following:

- Amazon ECS cluster and service
- Load balancer target group and listener information
- Deployment roll back strategy
- Traffic rerouting settings
- Original revision termination settings
- Deployment configuration
- CloudWatch alarms configuration that can be set up to stop deployments
- SNS or CloudWatch Events settings for notifications

For more information, see Working with Deployment Groups in the *AWS CodeDeploy User Guide*.

CodeDeploy deployment configuration

Specifies how CodeDeploy routes production traffic to your replacement task set during a deployment. The following pre-defined linear and canary deployment configuration are available. You can also create custom defined linear and canary deployments as well. For more information, see Working with Deployment Configurations in the *AWS CodeDeploy User Guide*.

Deployment configuration	Description
`CodeDeployDefault.ECSLinear10PercentEvery1Minutes`	Shifts 10 percent of traffic every minute until all traffic is shifted.
`CodeDeployDefault.ECSLinear10PercentEvery3Minutes`	Shifts 10 percent of traffic every three minutes until all traffic is shifted.
`CodeDeployDefault.ECSCanary10percent5Minutes`	Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed five minutes later.
`CodeDeployDefault.ECSCanary10percent15Minutes`	Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed 15 minutes later.
`CodeDeployDefault.ECSAllAtOnce`	Shifts all traffic to the updated Amazon ECS container at once.

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.
- Amazon ECS service auto scaling is not supported when using the blue/green deployment type. As a workaround, you can suspend scaling processes on the Amazon EC2 Auto Scaling groups created

for your service before the service deployment, then resume the processes once the deployment has completed. For more information, see Suspending and resuming scaling processes in the *Amazon EC2 Auto Scaling User Guide*.

- Cluster capacity providers are not supported when using the blue/green deployment type.
- Tasks using the Fargate launch type or the `CODE_DEPLOY` deployment controller types don't support the `DAEMON` scheduling strategy.
- When you initially create a CodeDeploy application and deployment group, you must specify the following:
  - You must define two target groups for the load balancer. One target group should be the initial target group defined for the load balancer when the Amazon ECS service was created. The second target group's only requirement is that it can't be associated with a different load balancer than the one the service uses.
- When you create a CodeDeploy deployment for an Amazon ECS service, CodeDeploy creates a *replacement task set* (or *green task set*) in the deployment. If you added a test listener to the load balancer, CodeDeploy routes your test traffic to the replacement task set. This is when you can run any validation tests. Then CodeDeploy reroutes the production traffic from the original task set to the replacement task set according to the traffic rerouting settings for the deployment group.

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. 352).

To update an existing Amazon ECS service that is using the blue/green deployment type, see Updating a Service (p. 363).

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.ECSAllAtOnce` by

Resource	Default Setting
	default, which routes all traffic at one time from the blue task set to the green task set. The deployment configuration can be changed using the AWS CodeDeploy console after the service is created.
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 a CodeDeploy deployment group in the CodeDeploy console, see Change Deployment Group Settings with CodeDeploy in the *AWS CodeDeploy User Guide*.

A tutorial walking through the steps needed to create a service using the blue/green deployment type is provided. For more information, see Tutorial: Creating a Service Using a Blue/Green Deployment (p. 671).

Support for performing a blue/green deployment has been added for AWS CloudFormation. For more information, see Perform Amazon ECS blue/green deployments through CodeDeploy using AWS CloudFormation in the *AWS CloudFormation User Guide*.

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. 465).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy:CreateApplication",
        "codedeploy:CreateDeployment",
        "codedeploy:CreateDeploymentGroup",
        "codedeploy:GetApplication",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:ListApplications",
        "codedeploy:ListDeploymentGroups",
        "codedeploy:ListDeployments",
        "codedeploy:StopDeployment",
        "codedeploy:GetDeploymentTarget",
        "codedeploy:ListDeploymentTargets",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision",
```

```
        "codedeploy:BatchGetApplicationRevisions",
        "codedeploy:BatchGetDeploymentGroups",
        "codedeploy:BatchGetDeployments",
        "codedeploy:BatchGetApplications",
        "codedeploy:ListApplicationRevisions",
        "codedeploy:ListDeploymentConfigs",
        "codedeploy:ContinueDeployment",
        "sns:ListTopics",
        "cloudwatch:DescribeAlarms",
        "lambda:ListFunctions"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Note
In addition to the standard Amazon ECS permissions required to run tasks and services, 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. 493).

The Create Service Example (p. 462) and Update Service Example (p. 463) 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.

External Deployment Considerations

Consider the following when using the external deployment type:

- Service auto scaling is not supported when using an external deployment controller.

- If using a load balancer for the task task, the supported load balancer types are either an Application Load Balancer or a Network Load Balancer.
- Tasks using the Fargate launch type or `EXTERNAL` deployment controller types don't support the `DAEMON` scheduling strategy.

External Deployment Workflow

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`

   Type: String

   Required: Yes

   The name of your service. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a Region or across multiple Regions.

   `desiredCount`

   The number of instantiations of the specified task set task definition to place and keep running within the service.

   `deploymentConfiguration`

   Optional deployment parameters that control how many tasks run during a deployment and the ordering of stopping and starting tasks. For more information, see deploymentConfiguration.

   `tags`

   Type: Array of objects

   Required: No

   The metadata that you apply to the service to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see Tagging Your Amazon ECS Resources (p. 407).

   `key`

   Type: String

   Length Constraints: Minimum length of 1. Maximum length of 128.

   Required: No

   One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

   `value`

   Type: String

   Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

`enableECSManagedTags`

Specifies whether to enable Amazon ECS managed tags for the tasks within the service. For more information, see Tagging Your Resources for Billing (p. 409).

`propagateTags`

Type: String

Valid values: `TASK_DEFINITION` | `SERVICE`

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation, use the `TagResource` API action.

`healthCheckGracePeriodSeconds`

Type: Integer

Required: No

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 enters a `RUNNING` state. This is only valid if your service is configured to use a load balancer. If your service has a load balancer defined and you do not specify a health check grace period value, the default value of `0` is used.

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. 338).

`placementConstraints`

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task (this limit includes constraints in the task definition and those specified at run time). If you are using the Fargate launch type, task placement constraints aren't supported.

`placementStrategy`

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

The following is an example service definition for creating a service using an external deployment controller.

```
{
    "cluster": "",
    "serviceName": "",
    "desiredCount": 0,
    "role": "",
    "deploymentConfiguration": {
        "maximumPercent": 0,
        "minimumHealthyPercent": 0
    },
    "placementConstraints": [
        {
            "type": "distinctInstance",
            "expression": ""
        }
    ],
    "placementStrategy": [
        {
            "type": "binpack",
            "field": ""
        }
    ],
    "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`

   Type: String

   Valid values: `EC2` | `FARGATE`

   Required: No

   The launch type on which to run your service. If a launch type is not specified, `EC2` is used by default. For more information, see Amazon ECS Launch Types (p. 119).

   If a `launchType` is specified, the `capacityProviderStrategy` parameter must be omitted.

   `platformVersion`

   Type: String

   Required: No

The platform version on which your tasks in the service are running. A platform version is only specified for tasks using the Fargate launch type. If one is not specified, the latest version (`LATEST`) is used by default.

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. 33).

Note
Platform versions are not specified for tasks using the EC2 launch type.

`loadBalancers`

A load balancer object representing the load balancer to use with your service. When using an external deployment controller, only Application Load Balancers and Network Load Balancers are supported. If you're using an Application Load Balancer, only one Application Load Balancer target group is allowed per task set.

The following snippet shows an example loadBalancer object to use.

```
"loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
]
```

Note
When specifying a `loadBalancer` object, you must specify a `targetGroupArn` and omit the `loadBalancerName` parameters.

`networkConfiguration`

The network configuration for the service. This parameter is required for task definitions that use the `awsvpc` network mode to receive their own elastic network interface, and it's not supported for other network modes. For more information, see Task Networking with the `awsvpc` Network Mode (p. 141)..

`serviceRegistries`

The details of the service discovery registries to assign to this service. For more information, see Service Discovery (p. 403).

`scale`

A floating-point percentage of the desired number of tasks to place and keep running in the task set. The value is specified as a percent total of a service's `desiredCount`. Accepted values are numbers between 0 and 100.

The following is a JSON example for creating a task set for an external deployment controller.

```
{
    "service": "",
    "cluster": "",
    "externalId": "",
    "taskDefinition": "",
    "networkConfiguration": {
        "awsvpcConfiguration": {
```

```
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "scale": {
        "value": null,
        "unit": "PERCENT"
    },
    "clientToken": ""
}
```

3. When service changes are needed, use the `UpdateService`, `UpdateTaskSet`, or `CreateTaskSet` API action depending on which parameters you're updating. If you created a task set, use the `scale` parameter for each task set in a service to determine how many tasks to keep running in the service. For example, if you have a service that contains `tasksetA` and you create a `tasksetB`, you might test the validity of `tasksetB` before wanting to transition production traffic to it. You could set the `scale` for both task sets to `100`, and when you were ready to transition all production traffic to `tasksetB`, you could update the `scale` for `tasksetA` to `0` to scale it down.

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.

Amazon ECS services support the Application Load Balancer, Network Load Balancer, and Classic Load Balancer load balancer types. 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. 379).

Application Load Balancers offer several features that make them attractive for use with Amazon ECS services:

- Each service can serve traffic from multiple load balancers and expose multiple load balanced ports by specifying multiple target groups.
- They are supported by tasks using both the Fargate and EC2 launch types.
- 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

- Service Load Balancing Considerations (p. 377)
- Load Balancer Types (p. 379)
- Creating a Load Balancer (p. 382)
- Registering Multiple Target Groups with a Service (p. 393)

Service Load Balancing Considerations

Consider the following when you use service load balancing.

Application Load Balancer and Network Load Balancer Considerations

The following considerations are specific to Amazon ECS services using Application Load Balancers or Network Load Balancers:

- For services that use an Application Load Balancer or Network Load Balancer, you cannot attach more than five target groups to a service.
- For services with tasks using the `awsvpc` network mode, when you create a target group for your service, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.
- If your service using an Application Load Balancer requires access to multiple load balanced ports, such as port 80 and port 443 for an HTTP/HTTPS service, you can configure two listeners. One listener is responsible for HTTPS that forwards the request to the service, and another listener that is responsible for redirecting HTTP requests to the appropriate HTTPS port. For more information, see Create a Listener to Your Application Load Balancer in the *User Guide for Application Load Balancers*.
- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
- After you create a service, the 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 stopped and restarted. This process continues until your service reaches the number of desired running tasks.
- The Application Load Balancer slow start mode is supported. For more information, see Application Load Balancer Slow Start Mode Considerations (p. 378). about slow start mode, see Target Groups for Your Application Load Balancers.

- When using Network Load Balancers configured with IP addresses as targets, requests are seen as coming from the Network Load Balancers private IP address. This means that services behind an Network Load Balancer are effectively open to the world as soon as you allow incoming requesets and health checks in the target's security group.
- If you are experiencing problems with your load balancer-enabled services, see Troubleshooting service load balancers (p. 699).

Application Load Balancer Slow Start Mode Considerations

Application Load Balancers enabled for slow start mode are supported for Amazon ECS services. For more information about slow start mode, see Target Groups for Your Application Load Balancers.

To ensure that the service scheduler ignores unhealthy container health checks until your tasks have warmed up and are ready to receive traffic, the following configurations are required:

- You must configure your container health check to return an `UNHEALTHY` status until the slow start period has ended.
- You must configure the health check grace period value for your Amazon ECS service for the same duration as the slow start mode duration.

Consider the following when you use different task network modes with Application Load Balancer slow start mode:

- When using `awsvpc` network mode, each task is assigned its own elastic network interface (ENI) and IP address which allows the Application Load Balancer to register each task as a target in the target group. This enables each newly registered target to have slow start mode enabled.
- When using `host` network mode, the task bypasses the Docker networking constructs and maps container ports directly to the Amazon EC2 instance's network interface or interfaces. You register the container instance as the Application Load Balancer target as opposed to the IP address of the task. This means you can only run one task per instance if you want slow start mode to work effectively. If you were to update an existing task or service, or restart the container instance, this does not re-register the container instance as an Application Load Balancer target, which would not cause the slow start duration to begin.
- When using `bridge` network mode, similarly to using `host` network mode, you register the container instance as the Application Load Balancer target as opposed to the Amazon ECS task so the same considerations described above apply.

Additionally, the following considerations are specific for using Application Load Balancer slow start mode and adding Amazon ECS tasks as targets:

- When you enable slow start for a target group, the targets already registered with the target group do not enter slow start mode.
- When you enable slow start for an empty target group and then register one or more targets using a single registration operation, these targets do not enter slow start mode. Newly registered targets enter slow start mode only when there is at least one registered target that is not in slow start mode.
- If you deregister a target in slow start mode, the target exits slow start mode. If you register the same target again, it enters slow start mode again.
- If a target in slow start mode becomes unhealthy and then healthy again before the duration period elapses, the target remains in slow start mode until the duration period elapses and then exits slow start mode. If a target that is not in slow start mode changes from unhealthy to healthy, it does not enter slow start mode.

Classic Load Balancer Considerations

The following considerations are specific to Amazon ECS services using Classic Load Balancers:

- Services with tasks that use the `awsvpc` network mode, such as those with the Fargate launch type, do not support Classic Load Balancers.
- 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 with host and port combinations, this configuration can cause issues if a task from one service stops. In this scenario, a task from one service stopping can cause 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 we recommend using an Application Load Balancer.

Load Balancer Types

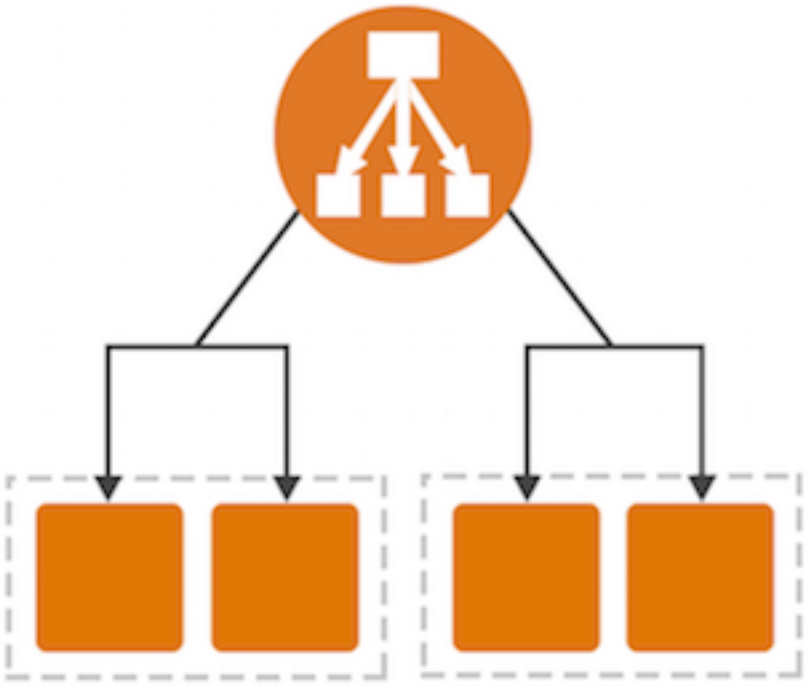
Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. Amazon ECS services can use either type of load balancer. Application Load Balancers are used to route HTTP/HTTPS (or Layer 7) traffic. Network Load Balancers and Classic Load Balancers are used to route TCP (or Layer 4) traffic.

Topics

- Application Load Balancer (p. 379)
- Network Load Balancer (p. 380)
- Classic Load Balancer (p. 381)

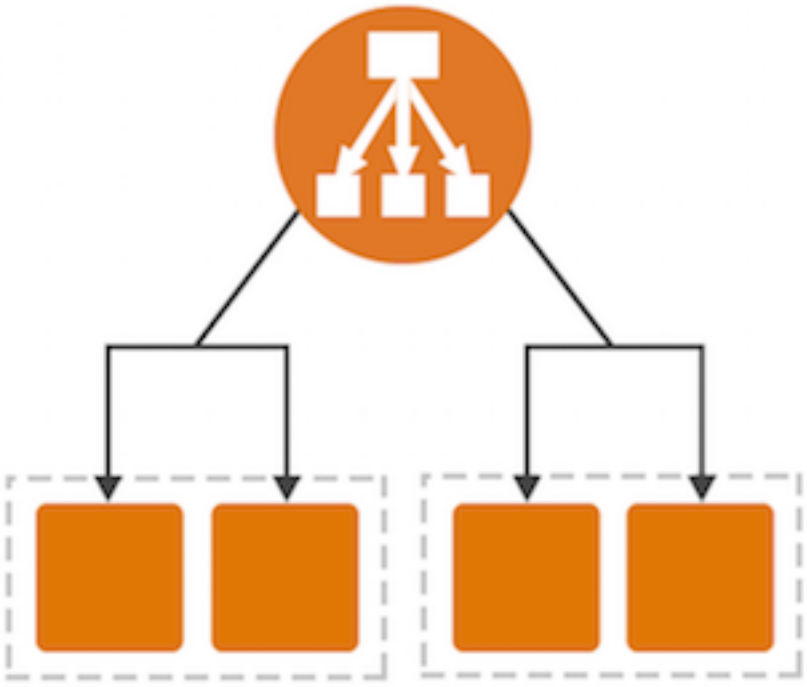
Application Load Balancer

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.



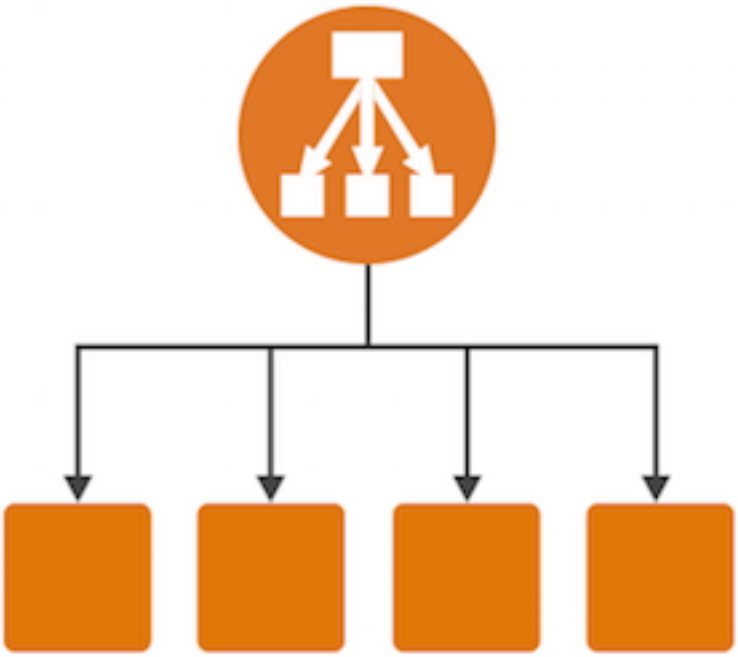
Network Load Balancer

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.



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.

Prior to using a load balancer with your Amazon ECS service, your account must already have the Amazon ECS service role created. For more information, see Creating the Service Role for Your Account (p. 383).

Topics

- Creating the Service Role for Your Account (p. 383)
- Creating an Application Load Balancer (p. 383)
- Creating a Network Load Balancer (p. 387)
- Creating a Classic Load Balancer (p. 389)

Creating 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 Service Scheduler IAM Role (p. 479) 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**.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

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**.

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 in the *User Guide for Application Load Balancers*.
5. Choose **Next: Configure Security Groups**.

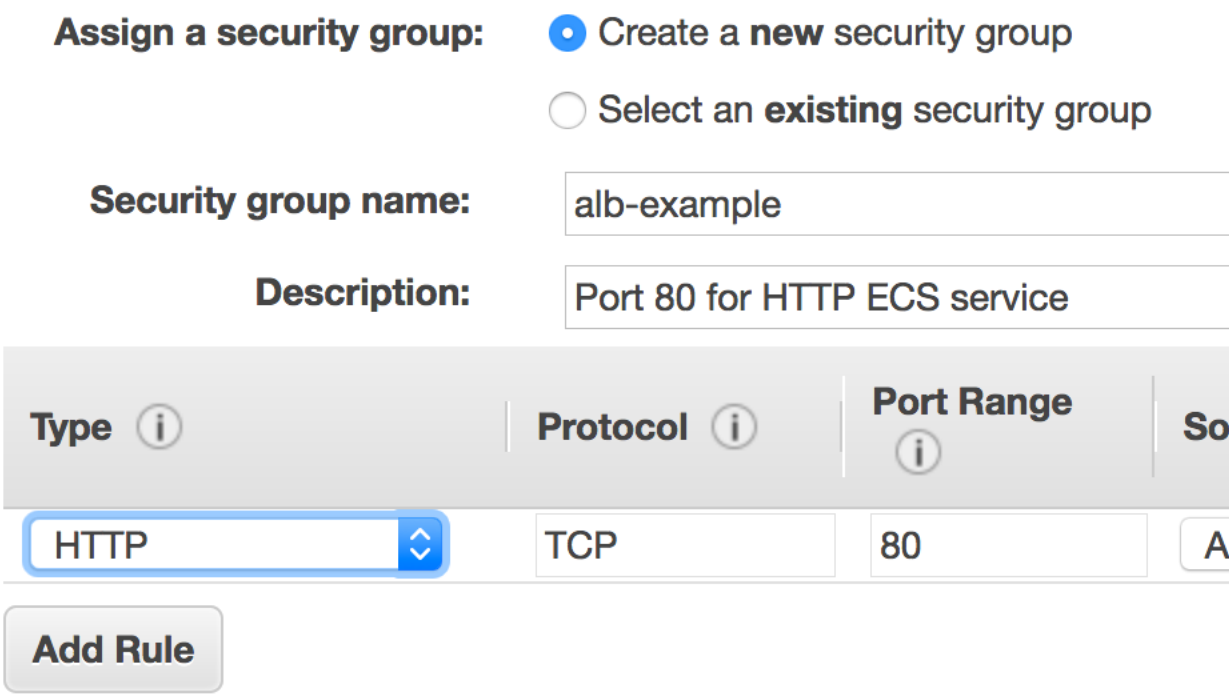
Configure Security Groups

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.



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.

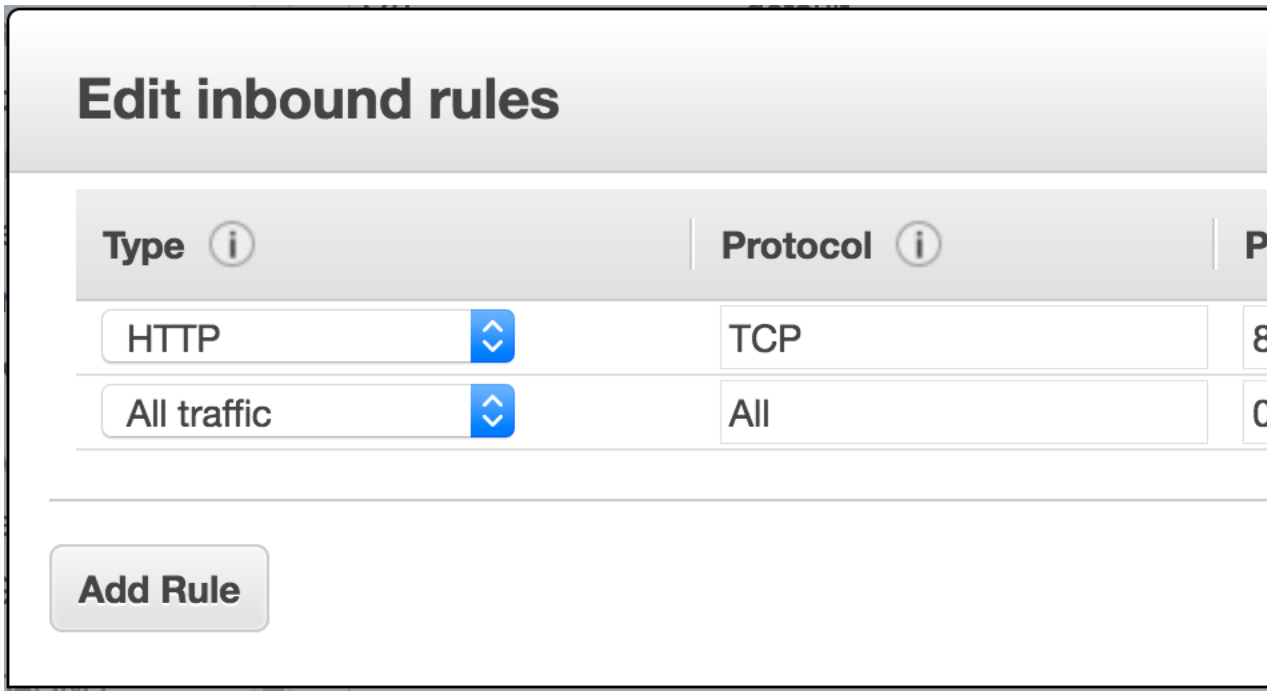
Create a Security Group Rule for Your Container Instances

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. 385). 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. 352).

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.
    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 in the *User Guide for 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. 352).

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.

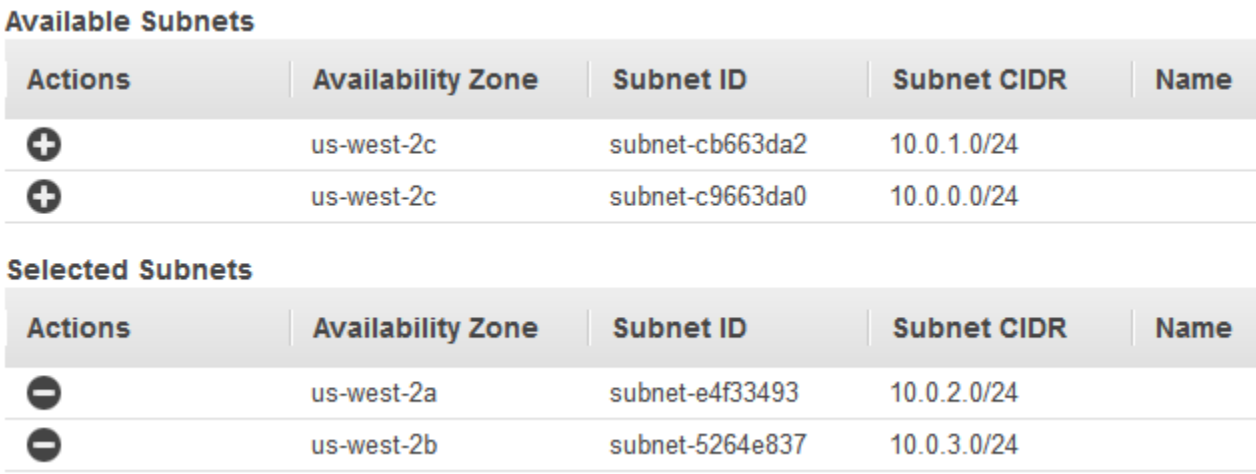
A *listener* is a process that checks for connection requests. It is configured with a protocol and port for the frontend (client to load balancer) connections and a protocol, and a protocol and port for the backend (load balancer to backend instance) connections. In this example, you configure a listener 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**.



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

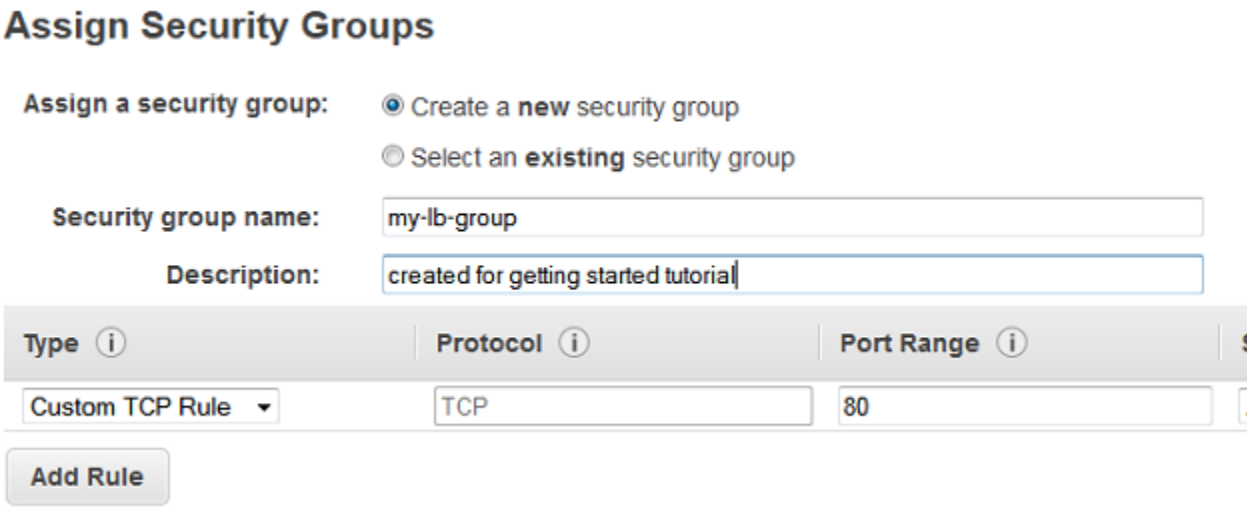
If you created your load balancer in a VPC, you must assign it a security group that allows inbound traffic to the ports that you specified for your load balancer and the health checks for your load balancer. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing 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.

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*.

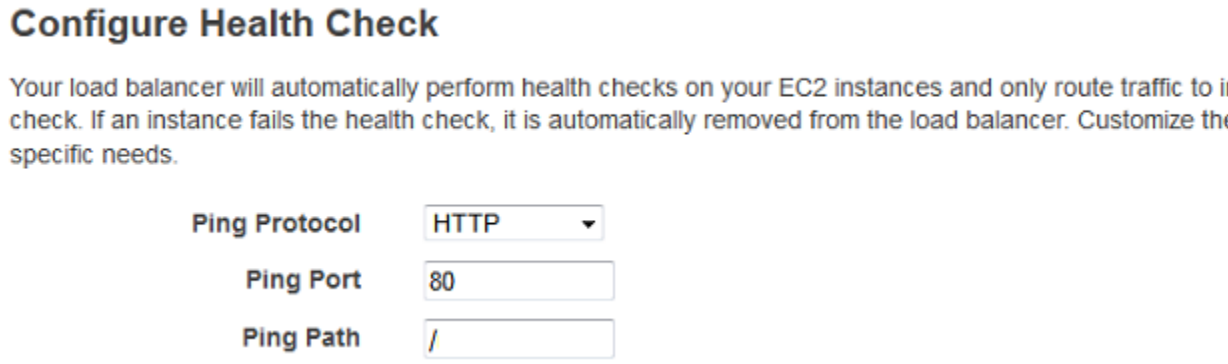
Configure Health Checks for Your Amazon EC2 Instances

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 Amazon EC2 instance hosting that task and reroutes the traffic to a healthy instance.

> **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.



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

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. 352).

Registering Multiple Target Groups with a Service

Your Amazon ECS service can serve traffic from multiple load balancers and expose multiple load balanced ports when you specify multiple target groups in a service definition.

To create a service specifying multiple target groups, you must create the service using the Amazon ECS API, SDK, AWS CLI, or an AWS CloudFormation template. After the service is created, you can view the service and the target groups registered to it with the AWS Management Console. It is not possible to update the load balancing configuration of an existing service.

Multiple target groups can be specified in a service definition using the following format. For the full syntax of a service definition, see Service Definition Template (p. 351).

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"container_name",
      "containerPort":container_port
   },
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"container_name",
      "containerPort":container_port
   }
]
```

Multiple Target Group Considerations

The following should be considered when you specify multiple target groups in a service definition:

- Multiple target groups are only supported when you use the Application Load Balancer or Network Load Balancer load balancer types.
- Multiple target groups are only supported when the service uses the rolling update (`ECS`) deployment controller type. If you are using the CodeDeploy or an external deployment controller, multiple target groups are not supported.
- Multiple target groups are supported for services containing tasks using both the Fargate and EC2 launch types.
- When creating a service that specifies multiple target groups, the Amazon ECS service-linked role must be created. The role is created by omitting the `role` parameter in API requests, or the `Role` property in AWS CloudFormation. For more information, see Service-Linked Role for Amazon ECS (p. 473).

Example Service Definitions

Following are a few example use cases for specifying multiple target groups in a service definition. For the full syntax of a service definition, see Service Definition Template (p. 351).

Example: Having separate load balancers for internal and external traffic

In the following use case, a service uses two separate load balancers, one for internal traffic and a second for internet-facing traffic, for the same container and port.

```
"loadBalancers":[
   //Internal ELB
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"nginx",
      "containerPort":8080
   },
   //Internet-facing ELB
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"nginx",
      "containerPort":8080
   }
]
```

Example: Exposing multiple ports from the same container

In the following use case, a service uses one load balancer but exposes multiple ports from the same container. For example, a Jenkins container might expose port 8080 for the Jenkins web interface and port 50000 for the API.

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"jenkins",
      "containerPort":8080
   },
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"jenkins",
      "containerPort":50000
   }
]
```

Example: Exposing ports from multiple containers

In the following use case, a service uses one load balancer and two target groups to expose ports from separate containers.

```
"loadBalancers":[
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_1/1234567890123456",
      "containerName":"webserver",
      "containerPort":80
   },
```

```
   {

 "targetGroupArn":"arn:aws:elasticloadbalancing:region:123456789012:targetgroup/
target_group_name_2/6543210987654321",
      "containerName":"database",
      "containerPort":3306
   }
]
```

Service Auto Scaling

Automatic scaling is the ability to increase or decrease the desired count of tasks in your Amazon ECS service automatically. Amazon ECS leverages the Application Auto Scaling service to provide this functionality. For more information, see the Application Auto Scaling User Guide.

Amazon ECS publishes CloudWatch metrics with your service's average CPU and memory usage. For more information, see Service Utilization (p. 421). You can use these and other CloudWatch metrics to scale out your service (add more tasks) to deal with high demand at peak times, and to scale in your service (run fewer tasks) to reduce costs during periods of low utilization.

Amazon ECS Service Auto Scaling supports the following types of automatic scaling:

- Target Tracking Scaling Policies (p. 396)—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. 401)—Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.
- Scheduled Scaling—Increase or decrease the number of tasks that your service runs based on the date and time.

IAM Permissions Required for Service Auto Scaling

Service Auto Scaling is made possible by a combination of the Amazon ECS, CloudWatch, and Application Auto Scaling APIs. Services are created and updated with Amazon ECS, alarms are created with CloudWatch, and scaling policies are created with Application Auto Scaling.

In addition to the standard IAM permissions for creating and updating services, the IAM user that accesses Service Auto Scaling settings must have the appropriate permissions for the services that support dynamic scaling. IAM users must have permissions to use the actions shown in the following example policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:*",
        "ecs:DescribeServices",
        "ecs:UpdateService",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DeleteAlarms",
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
```

```
        "cloudwatch:DescribeAlarmsForMetric",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DisableAlarmActions",
        "cloudwatch:EnableAlarmActions",
        "iam:CreateServiceLinkedRole",
        "sns:CreateTopic",
        "sns:Subscribe",
        "sns:Get*",
        "sns:List*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The Create Service Example (p. 462) and Update Service Example (p. 463) 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 also needs permission to describe your Amazon ECS services and CloudWatch alarms, and permissions to modify your service's desired count on your behalf. If you enable automatic scaling for your ECS services, it creates a service-linked role named `AWSServiceRoleForApplicationAutoScaling_ECSService`. This service-linked role grants Application Auto Scaling permission to describe the alarms for your policies, to monitor the current running task count of the service, and to modify the desired count of the service. The original managed Amazon ECS role for Application Auto Scaling was `ecsAutoscaleRole`, but it is no longer required. The service-linked role is the default role for Application Auto Scaling. For more information, see Service-Linked Roles in the *Application Auto Scaling User Guide*.

Note that if you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, you might need to add the `ecs:StartTelemetrySession` permission. For more information, see Enabling CloudWatch Metrics (p. 416).

Target Tracking Scaling Policies

With target tracking scaling policies, you select a metric and set a target value. Amazon ECS Service Auto Scaling creates and manages the CloudWatch alarms that 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.

Considerations

Keep the following considerations in mind:

- A target tracking scaling policy assumes that it should perform scale out when the specified metric is above the target value. You cannot use a target tracking scaling policy to scale out when the specified metric is below the target value.
- A target tracking scaling policy does not perform scaling when the specified metric has insufficient data. It does not perform scale in because it does not interpret insufficient data as low utilization.
- You may see gaps between the target value and the actual metric data points. This is because Service Auto Scaling always acts conservatively by rounding up or down when it determines how much

capacity to add or remove. This prevents it from adding insufficient capacity or removing too much capacity.

- To ensure application availability, the service scales out proportionally to the metric as fast as it can, but scales in more gradually.
- You can have multiple target tracking scaling policies for an Amazon ECS service, provided that each of them uses a different metric. The intention of Service Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the service if any of the target tracking policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion enabled) are ready to scale in.
- Do not edit or delete the CloudWatch alarms that Service Auto Scaling manages for a target tracking scaling policy. Service Auto Scaling deletes the alarms automatically when you delete the scaling policy.

Tutorial: Service Auto Scaling with Target Tracking

The following procedures help you to create an Amazon ECS cluster and a service that uses target tracking to scale out (and in) automatically based on demand.

In this tutorial, you use the Amazon ECS first-run wizard to create a cluster and a service that runs behind an Elastic Load Balancing load balancer. Then you configure a target tracking scaling policy that scales your service automatically based on the current application load as measured by the service's CPU utilization (from the **ECS, ClusterName, ServiceName** category in CloudWatch).

When the average 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), a scale-out alarm triggers Service Auto Scaling to add another task to your service to help out with the increased load. Conversely, when the average CPU utilization of your service drops below the target utilization for a sustained period of time, a 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 are using administrator credentials, and that you have 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).

Step 1: Create a Cluster and a Service

Start by creating a cluster and service using the Amazon ECS first-run wizard. The first-run wizard takes care of creating the necessary IAM roles 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).

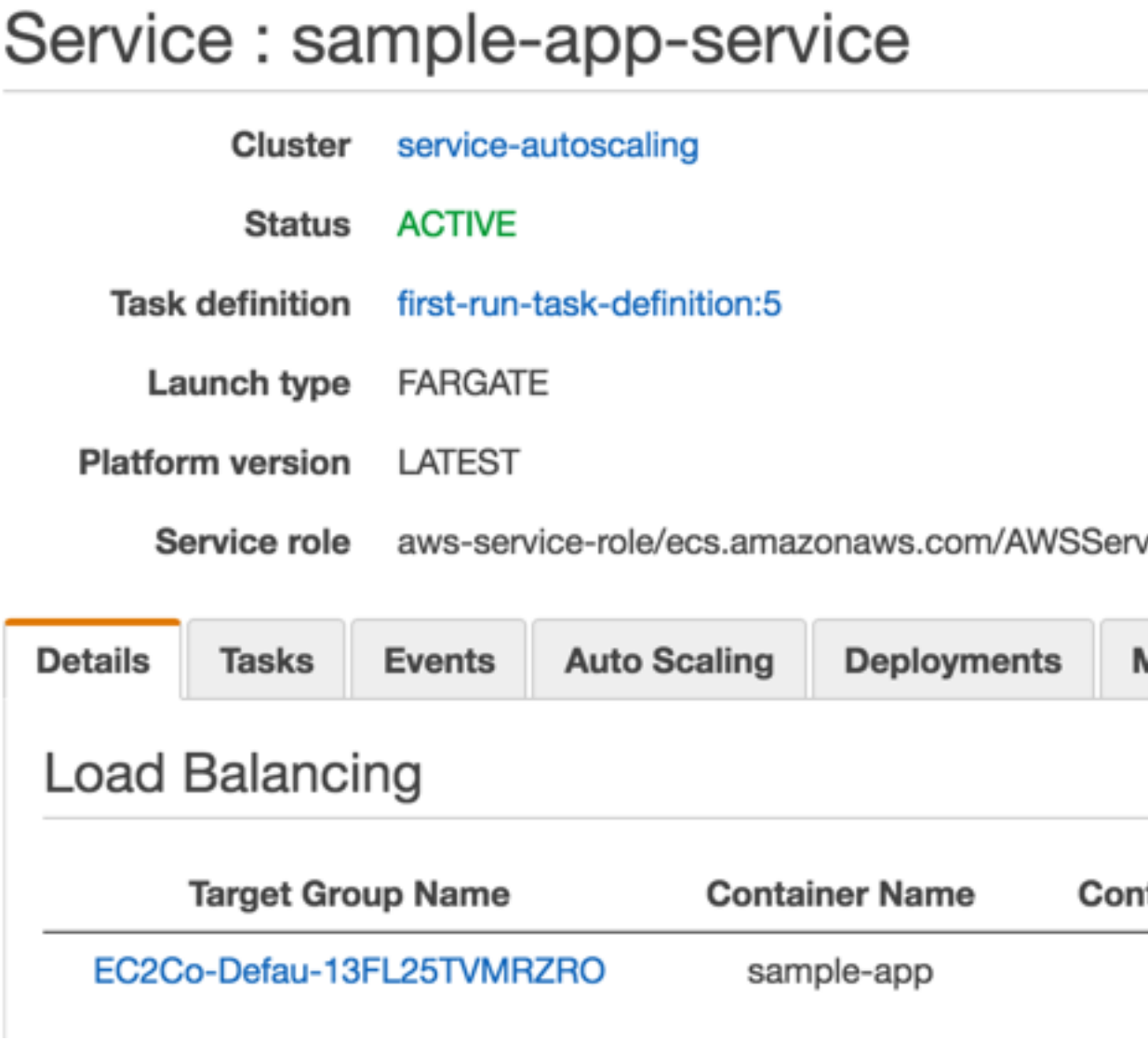
8. When your cluster and service are created, choose **View service**.

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 enable Service Auto Scaling by creating a target tracking scaling policy.

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 the `ecsAutoscaleRole`. If this role does not exist, choose **Create new role** to have the console create it for you.

To configure a target tracking scaling policy for 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` seconds. A scale-out activity increases the number of your service's tasks. While the scale-out cooldown period is in effect, the capacity that has been added by the previous scale-out activity that initiated the cooldown is calculated as part of the desired capacity for the next scale out. The intention is to continuously (but not excessively) scale out.
   f. For **Scale-in cooldown period**, enter `60` seconds. A scale-in activity reduces the number of your service's tasks. The scale-in cooldown period is used to block subsequent scale-in requests until it has expired. The intention is to scale in conservatively to protect your application's availability. However, if another alarm triggers a scale out activity during the cooldown period after a scale-in, Service Auto Scaling scales out your scalable target immediately.
   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 75% 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:

   ```
   $ sudo yum install -y httpd24-tools
   ```

   Run the following command, substituting your load balancer's DNS name.

   ```
   $ ab -n 100000 -c 1000 http://EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-east-1.elb.amazonaws.com/
   ```
3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

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: Next Steps

Go to the next step if you would like to delete the basic infrastructure that you just created for this tutorial. Otherwise, you can use this infrastructure as your base and try one or more of the following:

- To view these scaling activities from the Amazon ECS console, choose the **Events** tab of the service. When scaling events occur, you see informational messages here. For example:

```
Message: Successfully set desired count to 1. Change successfully fulfilled by ecs.
 Cause: monitor alarm TargetTracking-service/service-autoscaling/sample-webapp-AlarmLow-
fcd80aef-5161-4890-aeb4-35dde11ff42c in state ALARM triggered policy TargetTrackingPolicy
```

- If you have CloudWatch Container Insights set up and it's collecting Amazon ECS metrics, you can view metric data on the CloudWatch automatic dashboards. For more information, see Introducing Amazon CloudWatch Container Insights for Amazon ECS in the *AWS Compute Blog*.
- Learn how to set up CloudWatch Container Insights. Additional charges apply. For more information, see Amazon ECS CloudWatch Container Insights (p. 439) and Updating Cluster Settings (p. 71).

Step 5: Cleaning Up

When you have completed this tutorial, you may choose to keep your cluster, Auto Scaling group, load balancer, target tracking scaling policy, 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

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.

Step Scaling Policies

Although Amazon ECS Service Auto Scaling supports using Application Auto Scaling step scaling policies, we recommend using target tracking scaling policies instead. For example, if you want to scale your service when CPU utilization falls below or rises above a certain level, create a target tracking scaling policy based on the CPU utilization metric provided by Amazon ECS. For more information, see Target Tracking Scaling Policies (p. 396).

With step scaling policies, you create and manage the CloudWatch alarms that trigger the scaling process. If the target tracking alarms don't work for your use case, you can use step scaling. You can also use target tracking scaling with step scaling for an advanced scaling policy configuration. For example, you can configure a more aggressive response when utilization reaches a certain level.

Service Auto Scaling Considerations

- 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.
- The scaling policies that you create for Amazon ECS services support a cooldown period. This is the number of seconds after a scaling activity completes where previous scaling policy-related scaling activities can influence future scaling activities.
  - For scale-out policies, while the cooldown period is in effect, the capacity that has been added by the previous scale-out activity that initiated the cooldown is calculated as part of the desired capacity for the next scale out. The intention is to continuously (but not excessively) scale out.
  - For scale-in policies, the cooldown period is used to block subsequent scale in requests until it has expired. The intention is to scale in conservatively to protect your application's availability. However, if another alarm triggers a scale-out policy during the cooldown period after a scale in, automatic scaling scales out your service immediately.
- 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, Service Auto Scaling scales the desired count up to the minimum capacity value and then continues to scale out as required, based on the scaling policy associated with the alarm. However, a scale-in activity does not adjust the desired count, because it is already below the minimum capacity value.
- If a service's desired count is set above its maximum capacity value, and an alarm triggers a scale in activity, Service Auto Scaling scales the desired count out to the maximum capacity value and then continues to scale in as required, based on the scaling policy associated with the alarm. However, a scale-out activity does not adjust the desired count, because it is already above the maximum capacity value.
- During scaling activities, the actual running task count in a service is the value that Service Auto Scaling uses as its starting point, as opposed to the desired count, 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.

Amazon ECS Console Experience

Service Auto Scaling is disabled by default. You can enable it by configuring scaling policies from the **Auto Scaling** tab of your services in the AWS Management Console for Amazon ECS.

For step-by-step guidance for working with scaling policies from the console, see Creating a service (p. 352) and Updating a Service (p. 363). For more information about step scaling and a walkthrough, see Automatic Scaling with Amazon ECS in the *AWS Compute Blog*. For a target tracking walkthrough, see Target Tracking Scaling Policies (p. 396).

When you configure scaling policies for a service in the Amazon ECS console, your service is automatically registered as a scalable target with Application Auto Scaling, and your scaling policies are automatically in force as soon as they're successfully created.

AWS CLI and SDK Experience

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.

To configure scaling policies for your ECS service using the AWS CLI

1. Register your ECS service as a scalable target using the register-scalable-target command.
2. Create a scaling policy using the put-scaling-policy command.
3. [Step scaling] Create an alarm that triggers the scaling policy using the put-metric-alarm command.

For more information about configuring scaling policies using the AWS CLI, see the Application Auto Scaling User Guide.

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*.

Service discovery is available in the following AWS Regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
Middle East (Bahrain)	me-south-1

Region Name	Region
South America (São Paulo)	sa-east-1

Service Discovery Concepts

Service discovery consists of the following components:

- **Service discovery namespace**: A logical group of service discovery services that share the same domain name, such as `example.com`.
- **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. 33).
- 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.
- 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.
- DNS records for a service discovery service can be queried within your VPC. They use the following format: `<service discovery service name>.<service discovery namespace>`. For more information, see Step 3: Verify Service Discovery (p. 667).
- When doing a DNS query on the service name, A records return a set of IP addresses that correspond to your tasks. SRV records return a set of IP addresses and ports per task.
- If you have eight or fewer healthy records, Route 53 responds to all DNS queries with all of the healthy records.
- When all records are unhealthy, Route 53 responds to DNS queries with up to eight unhealthy records.
- You can configure service discovery for an ECS service that is behind a load balancer, but service discovery traffic is always routed to the task and not the load balancer.
- Service discovery does not support the use of Classic Load Balancers.
- It is recommended to use container-level health checks managed by Amazon ECS for your service discovery service.
  - **HealthCheckCustomConfig**—Amazon ECS manages health checks on your behalf. Amazon ECS uses information from container and health checks, as well as your task state, to update the health with AWS Cloud Map. This is specified using the `--health-check-custom-config` parameter when creating your service discovery service. For more information, see HealthCheckCustomConfig in the *AWS Cloud Map API Reference*.
- If you are using the Amazon ECS console, the workflow creates one service discovery service per ECS service. It maps all of the task IP addresses as A records, or task IP addresses and port as SRV records.
- Service discovery can only be configured when first creating a service. Updating existing services to configure service discovery for the first time or change the current configuration is not supported.
- The AWS Cloud Map resources created when service discovery is used must be cleaned up manually. For more information, see Step 4: Clean Up (p. 670) in the Tutorial: Creating a Service Using Service Discovery (p. 662) topic.

Amazon ECS Console Experience

The service creation workflow in the Amazon ECS console supports service discovery. service discovery can only be configured when first creating a service. Updating existing services to configure service discovery for the first time or change the current configuration is not supported.

To create a new Amazon ECS service that uses service discovery, see Creating a service (p. 352).

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.

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 a maximum of 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. 697):

```
(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. 363).

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. 695).
- 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. 690).
- Insufficient disk space on your container instance to create the container. In this case, you also see `CannotCreateContainerError` in your stopped task errors (p. 690). For more information, see `CannotCreateContainerError: API error (500): devmapper` (p. 698).

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

- Tagging Your Amazon ECS Resources (p. 407)
- Amazon ECS Usage Reports (p. 413)

Tagging Your Amazon ECS Resources

To help you manage your Amazon ECS tasks, services, task sets, 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. 184).

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

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (Amazon ECS API, AWS CLI, AWS SDK)
Amazon ECS tasks	Yes	Yes, from the task definition.	Yes
Amazon ECS services	Yes	Yes, from either the task definition or the service to the tasks in the service.	Yes
Amazon ECS task sets	Yes	No	Yes
Amazon ECS task definitions	Yes	No	Yes
Amazon ECS clusters	Yes	No	Yes
Amazon ECS container instances	Yes	Yes, from the Amazon EC2 instance. For more information, see Adding Tags to a Container Instance (p. 410).	Yes

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 `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values as it is 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. 413).

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. 184).

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. 409)
- Adding and Deleting Tags on an Individual Resource (p. 410)
- Adding Tags to a Container Instance (p. 410)

Adding Tags on an Individual Resource During Launch

The following resources allow you to specify tags when you create the resource.

Task	Console
Run one or more tasks.	Running Tasks (p. 316)
Create a service.	Creating a service (p. 352)
Create a task set.	External Deployment (p. 371)
Register a task definition.	Creating a Task Definition (p. 75)
Create a cluster.	Creating a Cluster (p. 38)
Run one or more container instances.	Launching an Amazon ECS Container Instance (p. 219)

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. 410).

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. 223).

  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:

```
#!/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`:

```
#!/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

```
{
    "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

Task	AWS CLI	API Action
Add or overwrite one or more tags.	tag-resource	TagResource
Delete one or more tags.	untag-resource	UntagResource

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.

```
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.

```
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.

```
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.

Task	AWS CLI	AWS Tools for Windows PowerShell	API Action
Run one or more tasks.	run-task	Start-ECSTask	RunTask
Create a service.	create-service	New-ECSService	CreateService
Create a task set.	create-task-set	New-ECSTaskSet	CreateTaskSet
Register a task definition.	register-task-definition	Register-ECSTaskDefinition	RegisterTaskDefinition
Create a cluster.	create-cluster	New-ECSCluster	CreateCluster
Run one or more container instances.	run-instances	New-EC2Instance	RunInstances

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`.

```
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`.

```
aws ecs create-service --service-name application --task-definition task-def-app --tags
 key=stack,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 CPU usage as `vCPU-Hours` and memory usage as `GB-Hours` for each task that was run. How that data is presented depends on the launch type of the task, as described below.

For tasks using the Fargate launch type, the `lineItem/Operation` column will show `FargateTask` and you will see the cost associated with each task.

For tasks using the EC2 launch type, the `lineItem/Operation` column will show `ECSTask-EC2` and the tasks will not have a direct cost associated with them. The vCPU and/or memory usage shown in the report represents the total resources reserved by the task over the billing period indicated. These values can be used to determine the cost of your underlying cluster of Amazon EC2 instances. The cost and usage data for your Amazon EC2 instances will be listed separately under the Amazon EC2 service.

You can also use the Amazon ECS managed tags to identify the service or cluster that each task belongs to. For more information, see Tagging Your Resources for Billing (p. 409).

> **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

For more information about creating an AWS Cost and Usage Report, see AWS Cost and Usage Report in the *AWS Billing and Cost Management User Guide*.

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. 414)
- Amazon ECS CloudWatch Metrics (p. 416)
- Amazon ECS Events and EventBridge (p. 428)
- Amazon ECS CloudWatch Container Insights (p. 439)
- Logging Amazon ECS API Calls with AWS CloudTrail (p. 441)

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. 416).

  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. 424).
- 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. 143).

  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. 237).
- 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 Events and EventBridge (p. 428) in this guide and What Is Amazon CloudWatch Events? in the *Amazon CloudWatch Events User Guide*.
- 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. 441) in this guide, and Working with CloudTrail Log Files in the *AWS CloudTrail User Guide*.

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. 416)
- Available Metrics and Dimensions (p. 416)
- Cluster Reservation (p. 419)
- Cluster Utilization (p. 420)
- Service Utilization (p. 421)
- Service RUNNING Task Count (p. 422)
- Viewing Amazon ECS Metrics (p. 422)
- Tutorial: Scaling Container Instances with CloudWatch Alarms (p. 424)

Enabling CloudWatch Metrics

Any Amazon ECS 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 Amazon ECS task or service using the EC2 launch type, your Amazon ECS container instances require version 1.4.0 or later of the container agent to enable CloudWatch metrics. 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. 264).

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. 268).

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. 487).

> **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. 270).

Available Metrics and Dimensions

The following sections list the metrics and dimensions that Amazon ECS sends to Amazon CloudWatch.

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.

`CPUReservation`

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. Only container instances in `ACTIVE` or `DRAINING` status will affect CPU reservation metrics. 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.

`CPUUtilization`

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. Only container instances in `ACTIVE` or `DRAINING` status will affect CPU utilization metrics. 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.

Valid dimensions: `ClusterName`, `ServiceName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

`MemoryReservation`

The percentage of memory that is reserved by running tasks in the cluster.

Cluster memory reservation (this metric can only be filtered by `ClusterName`) 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. Only container instances in `ACTIVE` or `DRAINING` status will affect memory reservation metrics. 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.

`MemoryUtilization`

The percentage of memory that is used in the cluster or service.

Cluster memory utilization (metrics that are filtered by `ClusterName` without `ServiceName`) 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. Only container instances in `ACTIVE` or `DRAINING` status will affect memory utilization metrics. Cluster memory utilization metrics are only used for tasks using the EC2 launch type.

Service memory utilization (metrics that are filtered by `ClusterName` and `ServiceName`) 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.

Valid dimensions: `ClusterName`, `ServiceName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

`GPUReservation`

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. Only container instances in `ACTIVE` or `DRAINING` status will affect GPU reservation metrics.

Valid dimensions: `ClusterName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

Note
If you're using tasks with the EC2 launch type and have Linux container instances, the Amazon ECS container agent relies on Docker `stats` metrics to gather CPU and memory data for each container running on the instance. For burstable performance instances (T3, T3a, and T2 instances), the CPU utilization metric may reflect different data compared to instance-level CPU metrics.

Dimensions for Amazon ECS Metrics

Amazon ECS metrics use the `AWS/ECS` namespace and provide metrics for the following dimensions. Metrics for a dimension only reflect the resources with running tasks during a period. For example, if you

have a cluster with one service in it but that service has no tasks in a `RUNNING` state, there will be no metrics sent to CloudWatch. If you have two services and one of them has running tasks and the other doesn't, only the metrics for the service with running tasks would be sent.

`ClusterName`

This dimension filters the data that you request for all resources in a specified cluster. All Amazon ECS metrics are filtered by `ClusterName`.

`ServiceName`

This dimension filters the data that you request for all resources in a specified service within a specified cluster.

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. Only container instances in `ACTIVE` or `DRAINING` status will affect cluster reservation metrics. 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.

```
                                (Total CPU units reserved by tasks in cluster) x 100
Cluster CPU reservation =  ------------------------------------------------------------
                           (Total CPU units registered by container instances in cluster)
```

```
                                    (Total MiB of memory reserved by tasks in cluster x
 100)
Cluster memory reservation =
 ------------------------------------------------------------------
                              (Total MiB of memory registered by container instances in
 cluster)
```

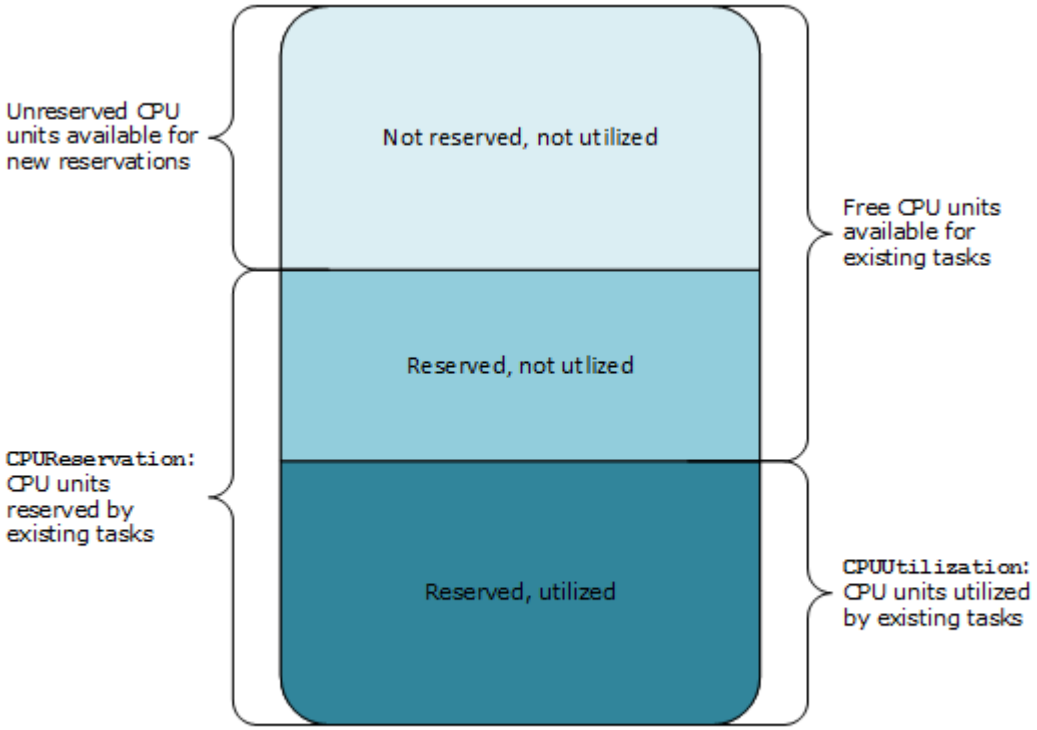
```
                                    (Total GPUs reserved by tasks in cluster x 100)
Cluster GPU reservation =
 ------------------------------------------------------------------
                              (Total GPUs registered by container instances in cluster)
```

When you run a task in a cluster, Amazon ECS parses its task definition and reserves the aggregate CPU units, MiB of memory, and GPUs that are specified in its container definitions. Each minute, Amazon ECS calculates the number of CPU units, MiB of memory, and GPUs that are currently reserved for each task that is running in the cluster. The total amount of CPU, memory, and GPUs reserved for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster. If you specify a soft limit (`memoryReservation`), 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. 91) task definition parameter documentation.



Aggregate cluster CPU units

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. Only container instances in `ACTIVE` or `DRAINING` status will affect cluster utilization metrics. 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.

```
                                (Total CPU units used by tasks in cluster) x 100
Cluster CPU utilization =  ------------------------------------------------------------
                          (Total CPU units registered by container instances in cluster)
```

```
                                   (Total MiB of memory used by tasks in cluster x 100)
Cluster memory utilization =
 ------------------------------------------------------------------
                             (Total MiB of memory registered by container instances in
 cluster)
```

Each minute, the Amazon ECS container agent on each 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 `c4.4xlarge` instance and a `c4.large` instance. The `c4.4xlarge` instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The `c4.large` instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If ten tasks are running on this cluster and each task consumes 1,024 CPU units and 2,048 MiB of memory, a total of 10,240 CPU units and 20,480 MiB of memory are used on the cluster. This is reported to CloudWatch as 55% CPU utilization and 60% memory utilization for the cluster.

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.

```
                                    (Total CPU units used by tasks in service) x 100
Service CPU utilization =
 ---------------------------------------------------------------------------
                          (Total CPU units specified in task definition) x (number of
 tasks in service)
```

```
                                       (Total MiB of memory used by tasks in service) x
 100
Service memory utilization =
 ---------------------------------------------------------------------------------
                             (Total MiB of memory specified in task definition) x (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.

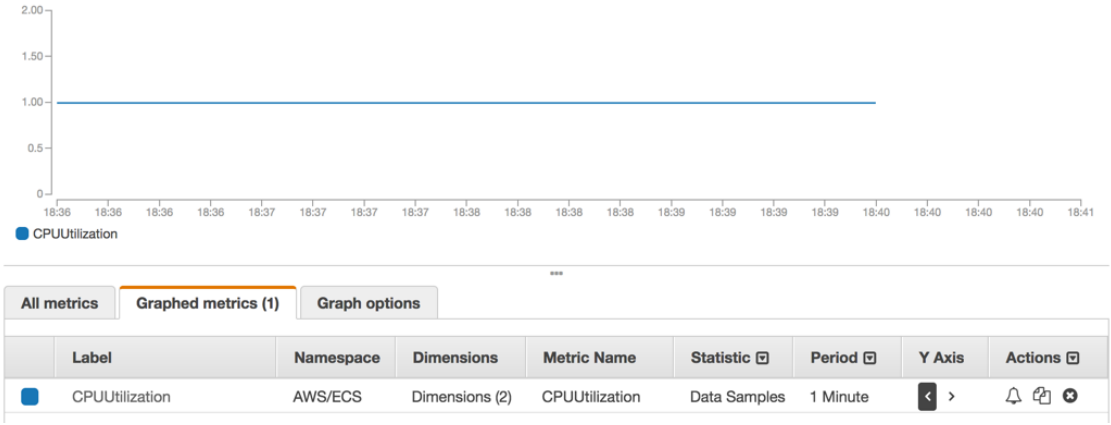
Service RUNNING Task Count

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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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. 422)
- Viewing Service Metrics on the Amazon ECS Console (p. 423)
- Viewing Amazon ECS Metrics on the CloudWatch Console (p. 423)

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. 419) and Cluster Utilization (p. 420).

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.
3. On the **Cluster: *`cluster-name`*** page, choose **Metrics**.

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. 421).

To view service metrics in the console

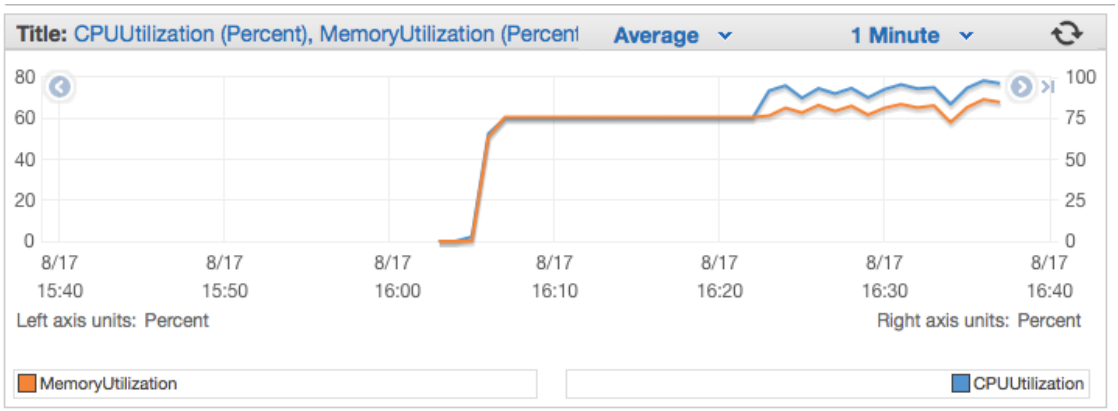
1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Select the cluster that contains the service that you want to view metrics for.
3. On the **Cluster: *`cluster-name`*** page, choose **Services**.
4. Choose the service that you want to view metrics for.
5. On the **Service: *`service-name`*** 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. 419), Cluster Utilization (p. 420), Service Utilization (p. 421), and the Service `RUNNING` Task Count (p. 422). For more information, see the Amazon CloudWatch User Guide.

To view metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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.



Tutorial: Scaling Container Instances with CloudWatch Alarms

Note
In December 2019, Amazon ECS launched cluster auto scaling, as an alternative method for scaling container instances. For more information, see Amazon ECS Cluster Auto Scaling (p. 42).

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 out (and in) 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 and a step scaling policy 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 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. For more information, see Enabling CloudWatch Metrics (p. 416).

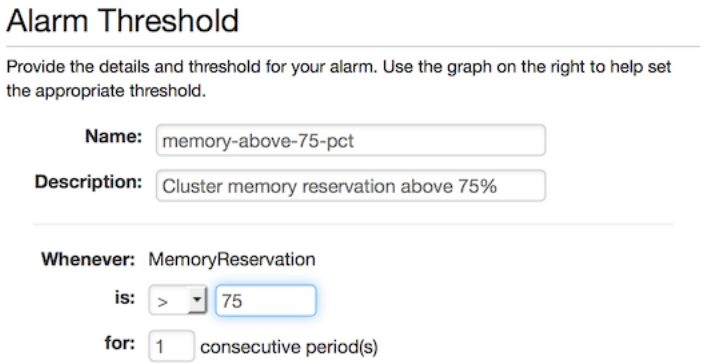
Step 1: Create a CloudWatch Alarm for a Metric

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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
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 Launch Configurations in 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/.
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 latest Amazon ECS-optimized Amazon Linux 2 AMI for your Auto Scaling group. For information on how to retrieve the latest Amazon ECS-optimized Amazon Linux 2 AMI, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 211).
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. 486).
   - **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. Expand the **Advanced Details** section to specify user data for your Amazon ECS container instances. For more information, see Amazon ECS Container Agent Configuration (p. 270).

Paste the following script into the **User data** field. Reference the cluster name that you are working with.

```
#!/bin/bash
echo ECS_CLUSTER=my-cluster >> /etc/ecs/ecs.config
```

10. Choose **Next: Add Storage**.
11. On the **Add Storage** step of the **Create Auto Scaling Group** wizard, make any storage configuration changes needed for your instances and choose **Next: Configure Security Group**.
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 with Step Scaling Policies

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 with step scaling policies

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.
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. Choose **Scale the Auto Scaling group using step or simple scaling policies**.
5. 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 capacity units (instances) to add to your cluster when the alarm is triggered.
6. 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.

7. 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 see your instances launching in the Amazon EC2 console **Instances** page. These instances should register into your ECS cluster as well after they launch.

Verify that the EC2 instances are registered with the cluster. From the ECS console, select the cluster that you registered your instances with. On the **Cluster** page, choose **ECS Instances**. Verify that the **Agent Connected** value is **True** for the instances displayed.

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

After you no longer need a step scaling policy, you can delete it. You also need to delete the CloudWatch alarms. Deleting a step scaling policy deletes the underlying alarm action, but does not delete the CloudWatch alarm associated with the scaling policy, even if it no longer has an associated action.

To delete a step scaling policy and its associated CloudWatch alarm

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. Select the Auto Scaling group.
4. On the **Scaling Policies** tab, choose **Actions**, **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.
6. Do the following to delete the CloudWatch alarm that was associated with the policy.
    a. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
    b. On the navigation pane, choose **Alarms**.
    c. Choose the alarm and choose **Action**, **Delete**.
    d. When prompted for confirmation, choose **Delete**.

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/.
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 Events and EventBridge

Amazon EventBridge enables you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Adding events to log groups in CloudWatch Logs
- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon Simple Queue Service (Amazon SQS) queue

For more information, see Getting Started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

You can use Amazon ECS events for EventBridge to receive near real-time notifications regarding the current state of your Amazon ECS clusters. If your tasks are using the 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. For services, you can see events related to the health of your service.

Using EventBridge, you can build custom schedulers on top of Amazon ECS that are responsible for orchestrating tasks across clusters and monitoring the state of clusters in near real time. You can eliminate scheduling and monitoring code that continuously polls the Amazon ECS service for status changes and instead handle Amazon ECS state changes asynchronously using any EventBridge target. Targets might include AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, or Amazon Kinesis Data Streams.

An Amazon ECS event stream ensures that every event is delivered at least one time. If duplicate events are sent, the event provides enough information to identify duplicates. For more information, see Handling Events (p. 437).

Events are relatively ordered, so that you can easily tell when an event occurred in relation to other events.

Topics

- Amazon ECS Events (p. 428)
- Handling Events (p. 437)

Amazon ECS Events

Amazon ECS sends three types of events to EventBridge: container instance state change events, task state change events, and service action events. If these resources change, an event is triggered. 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.

Container state change and task state change events contain two `version` fields: one in the main body of the event, and one in the `detail` object of the event. The following describes the differences between these two fields:

- The `version` field in the main body of the event is set to `0` on all events. For more information about EventBridge parameters, see Events and Event Patterns in the *Amazon EventBridge User Guide*.
- The `version` field in the `detail` object of the event describes the version of the associated resource. Each time a resource changes state, this version is incremented. Because events can be sent multiple times, this field allows you to identify duplicate events. Duplicate events have the same version in the `detail` object. If you are replicating your Amazon ECS container instance and task state with EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the version reported in EventBridge for the resource (inside the `detail` object) to verify that the version in your event stream is current.

Service action events only contain the `version` field in the main body.

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 an EC2 instance was stopped.

> When you stop a container instance, the tasks that are running on it are transitioned to the `STOPPED` status.

The Amazon ECS container agent registers a container instance for the first time.

> The first time the Amazon ECS container agent registers a container instance (at launch or when first run manually), this creates a state change event for the instance.

The Amazon ECS container agent connects or disconnects from Amazon ECS.

> When the Amazon ECS container agent connects or disconnects from the Amazon ECS backend, it changes the `agentConnected` status of the container instance.

Note
The Amazon ECS container agent disconnects and reconnects several times per hour as a part of its normal operation, so agent connection events should be expected. These events are not an indication that there is an issue with the container agent or your container instance.

You upgrade the Amazon ECS container agent on an instance.

The container instance detail contains an object for the container agent version. If you upgrade the agent, this version information changes and 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 in the *Amazon Elastic Container Service API Reference*. For more information about EventBridge parameters, see Events and Event Patterns in the *Amazon EventBridge User Guide*.

```
{
  "version": "0",
  "id": "8952ba83-7be2-4ab5-9c32-6687532d15a2",
  "detail-type": "ECS Container Instance State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2016-12-06T16:41:06Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315"
  ],
  "detail": {
    "agentConnected": true,
    "attributes": [
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
      },
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role-network-host"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"
      },
      {
        "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
      },
      {
        "name": "com.amazonaws.ecs.capability.privileged-container"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
      },
      {
        "name": "com.amazonaws.ecs.capability.ecr-auth"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.20"
      },
```

```
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.21"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.22"
      },
      {
        "name": "com.amazonaws.ecs.capability.docker-remote-api.1.23"
      },
      {
        "name": "com.amazonaws.ecs.capability.task-iam-role"
      }
    ],
    "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
    "containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/
b54a2a04-046f-4331-9d74-3f6d7f6ca315",
    "ec2InstanceId": "i-f3a8506b",
    "registeredResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 2048
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 3767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
          "22",
          "2376",
          "2375",
          "51678",
          "51679"
        ]
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "remainingResources": [
      {
        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 1988
      },
      {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 767
      },
      {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
          "22",
          "2376",
          "2375",
          "51678",
          "51679"
        ]
```

```
      },
      {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
      }
    ],
    "status": "ACTIVE",
    "version": 14801,
    "versionInfo": {
      "agentHash": "aebcbca",
      "agentVersion": "1.13.0",
      "dockerVersion": "DockerVersion: 1.11.2"
    },
    "updatedAt": "2016-12-06T16:41:06.991Z"
  }
}
```

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. State changes might include changes from `PENDING` to `RUNNING` or from `RUNNING` to `STOPPED`.

You force deregistration of the underlying container instance with the `DeregisterContainerInstance` API operation and the `force` flag, either directly or with the AWS Management Console or SDKs.

Deregistering a container instance changes the status of the container instance and the connection status of the Amazon ECS container agent. If tasks are running on the container instance, the `force` flag must be set to allow deregistration. This stops all tasks on the instance.

The underlying container instance is stopped or terminated.

When you stop or terminate a container instance, the tasks that are running on it are transitioned to the `STOPPED` status.

A container in the task changes state.

The Amazon ECS container agent monitors the state of containers within tasks. For example, if a container that is running within a task stops, this container state change triggers an event.

A task using the Fargate Spot capacity provider receives a termination notice.

When a task is using the `FARGATE_SPOT` capacity provider and is stopped due to a Spot interruption, a task state change event is triggered.

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. If your containers are using an image hosted with Amazon ECR, the `imageDigest` field is returned.

Note
The values for the `createdAt`, `connectivityAt`, `pullStartedAt`, `startedAt`, `pullStoppedAt`, and `updatedAt` fields are UNIX timestamps in the response of a DescribeTasks action whereas in the task state change event they are ISO string timestamps.

For more information about CloudWatch Events parameters, see Events and Event Patterns in the *Amazon EventBridge User Guide*.

```
{
    "version": "0",
    "id": "3317b2af-7005-947d-b652-f55e762e571a",
    "detail-type": "ECS Task State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-01-23T17:57:58Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:task/FargateCluster/
c13b4cb40f1f4fe4a2971f76ae5a47ad"
    ],
    "detail": {
        "attachments": [
            {
                "id": "1789bcae-ddfb-4d10-8ebe-8ac87ddba5b8",
                "type": "eni",
                "status": "ATTACHED",
                "details": [
                    {
                        "name": "subnetId",
                        "value": "subnet-abcd1234"
                    },
                    {
                        "name": "networkInterfaceId",
                        "value": "eni-abcd1234"
                    },
                    {
                        "name": "macAddress",
                        "value": "0a:98:eb:a7:29:ba"
                    },
                    {
                        "name": "privateIPv4Address",
                        "value": "10.0.0.139"
                    }
                ]
            }
        ],
        "availabilityZone": "us-west-2c",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/FargateCluster",
        "containers": [
            {
                "containerArn": "arn:aws:ecs:us-west-2:111122223333:container/
cf159fd6-3e3f-4a9e-84f9-66cbe726af01",
                "lastStatus": "RUNNING",
                "name": "FargateApp",
                "image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/hello-
repository:latest",
                "imageDigest":
 "sha256:74b2c688c700ec95a93e478cdb959737c148df3fbf5ea706abe0318726e885e6",
                "runtimeId":
 "ad64cbc71c7fb31c55507ec24c9f77947132b03d48d9961115cf24f3b7307e1e",
                "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/FargateCluster/
c13b4cb40f1f4fe4a2971f76ae5a47ad",
```

```
                "networkInterfaces": [
                    {
                        "attachmentId": "1789bcae-ddfb-4d10-8ebe-8ac87ddba5b8",
                        "privateIpv4Address": "10.0.0.139"
                    }
                ],
                "cpu": "0"
            }
        ],
        "createdAt": "2020-01-23T17:57:34.402Z",
        "launchType": "FARGATE",
        "cpu": "256",
        "memory": "512",
        "desiredStatus": "RUNNING",
        "group": "family:sample-fargate",
        "lastStatus": "RUNNING",
        "overrides": {
            "containerOverrides": [
                {
                    "name": "FargateApp"
                }
            ]
        },
        "connectivity": "CONNECTED",
        "connectivityAt": "2020-01-23T17:57:38.453Z",
        "pullStartedAt": "2020-01-23T17:57:52.103Z",
        "startedAt": "2020-01-23T17:57:58.103Z",
        "pullStoppedAt": "2020-01-23T17:57:55.103Z",
        "updatedAt": "2020-01-23T17:57:58.103Z",
        "taskArn": "arn:aws:ecs:us-west-2:111122223333:task/FargateCluster/
c13b4cb40f1f4fe4a2971f76ae5a47ad",
        "taskDefinitionArn": "arn:aws:ecs:us-west-2:111122223333:task-definition/sample-
fargate:1",
        "version": 4,
        "platformVersion": "1.3.0"
    }
}
```


For a tutorial walkthrough of setting up a simple AWS Lambda function that listens for Amazon ECS task events and writes them out to a CloudWatch Logs log stream, see Tutorial: Listening for Amazon ECS CloudWatch Events (p. 680).

For a tutorial walkthrough of creating an SNS topic to email you when a task state change event occurs, see Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events (p. 682).

Service Action Events

Amazon ECS sends service action events with the detail type **ECS Service Action**. Unlike the container instance and task state change events, the service action events do not include a version number in the `details` response field. The following is an event pattern that is used to create an EventBridge rule for Amazon ECS service action events. For more information, see Creating an EventBridge Rule in the *Amazon EventBridge User Guide*.

```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Service Action"
    ]
}
```

Amazon ECS sends events with `INFO`, `WARN`, and `ERROR` event types. The following are the service action events.

Service Action Events with `INFO` Event Type

`SERVICE_STEADY_STATE`

The service is healthy and at the desired number of tasks, thus reaching a steady state.

`TASKSET_STEADY_STATE`

The task set is healthy and at the desired number of tasks, thus reaching a steady state.

`CAPACITY_PROVIDER_STEADY_STATE`

A capacity provider associated with a service reaches a steady state.

`SERVICE_DESIRED_COUNT_UPDATED`

When the service scheduler updates the computed desired count for a service or task set. This event is not sent when the desired count is manually updated by a user.

Service Action Events with `WARN` Event Type

`SERVICE_TASK_START_IMPAIRED`

The service is unable to consistently start tasks successfully.

`SERVICE_DISCOVERY_INSTANCE_UNHEALTHY`

A service using service discovery contains an unhealthy task. The service scheduler detects that a task within a service registry is unhealthy.

Service Action Events with `ERROR` Event Type

`SERVICE_DAEMON_PLACEMENT_CONSTRAINT_VIOLATED`

A task in a service using the `DAEMON` service scheduler strategy no longer meets the placement constraint strategy for the service.

`ECS_OPERATION_THROTTLED`

The service scheduler has been throttled due to the Amazon ECS API throttle limits.

`SERVICE_DISCOVERY_OPERATION_THROTTLED`

The service scheduler has been throttled due to the AWS Cloud Map API throttle limits. This can occur on services configured to use service discovery.

`SERVICE_TASK_PLACEMENT_FAILURE`

The service scheduler is unable to place a task. The cause will be described in the `reason` field.

A common cause for this service event being triggered is because of a lack of resources in the cluster to place the task. For example, not enough CPU or memory capacity on the available container instances or no container instances being available. Another common cause is when the Amazon ECS container agent is disconnected on the container instance, causing the scheduler to be unable to place the task.

`SERVICE_TASK_CONFIGURATION_FAILURE`

The service scheduler is unable to place a task due to a configuration error. The cause will be described in the `reason` field.

A common cause of this service event being triggered is because tags were being applied to the service but the user or role had not opted in to the new Amazon Resource Name (ARN) format in the Region. For more information, see Amazon Resource Names (ARNs) and IDs (p. 184). Another common cause is that Amazon ECS was unable to assume the task IAM role provided.

Example Service Steady State Event

Service steady state events are delivered in the following format. For more information about EventBridge parameters, see Events and Event Patterns in the *Amazon EventBridge User Guide*.

For a tutorial walkthrough of setting up a simple AWS Lambda function that listens for Amazon ECS service action events and writes them out to a CloudWatch Logs log stream, see Tutorial: Listening for Amazon ECS CloudWatch Events (p. 680).

For a tutorial walkthrough of creating an SNS topic to email you when a service event event occurs, see Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events (p. 682).

```
{
    "version": "0",
    "id": "af3c496d-f4a8-65d1-70f4-a69d52e9b584",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:27:22Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:27:22.695Z"
    }
}
```

Example Capacity Provider Steady State Event

Capacity provider steady state events are delivered in the following format.

```
{
    "version": "0",
    "id": "b9baa007-2f33-0eb1-5760-0d02a572d81f",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:37:00Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "CAPACITY_PROVIDER_STEADY_STATE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/ASG-tutorial-capacity-
provider"
        ],
        "createdAt": "2019-11-19T19:37:00.807Z"
    }
```

```
}
```

Example Service Task Start Impaired Event

Service task start impaired events are delivered in the following format.

```
{
    "version": "0",
    "id": "57c9506e-9d21-294c-d2fe-e8738da7e67d",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "WARN",
        "eventName": "SERVICE_TASK_START_IMPAIRED",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "createdAt": "2019-11-19T19:55:38.725Z"
    }
}
```

Example Service Task Placement Failure Event

Service task placement failure events are delivered in the following format. For more information about EventBridge parameters, see Events and Event Patterns in the *Amazon EventBridge User Guide*.

In the following example, the task was attempting to use the `FARGATE_SPOT` capacity provider but the service scheduler was unable to acquire any Fargate Spot capacity.

```
{
    "version": "0",
    "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
    "detail-type": "ECS Service Action",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2019-11-19T19:55:38Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "ERROR",
        "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
        "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "capacityProviderArns": [
            "arn:aws:ecs:us-west-2:111122223333:capacity-provider/FARGATE_SPOT"
        ],
        "reason": "RESOURCE:FARGATE",
        "createdAt": "2019-11-06T19:09:33.087Z"
    }
}
```

Handling Events

Amazon ECS sends events on an *at least once* basis. This means you may receive multiple copies of a given event. Additionally, events may not be delivered to your event listeners in the order in which the events occurred.

To enable proper ordering of events, the `detail` section of each event contains a `version` property. Each time a resource changes state, this `version` is incremented. Duplicate events have the same `version` in the `detail` object. If you are replicating your Amazon ECS container instance and task state with EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the `version` reported in EventBridge for the resource to verify that the version in your event stream is current. Events with a higher version property number should be treated as occurring later than events with lower version numbers.

Example: Handling Events in an AWS Lambda Function

The following example shows a Lambda function written in Python 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.

```
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
        )
```

Amazon ECS CloudWatch Container Insights

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. The metrics include utilization for resources such as CPU, memory, disk, and network. The metrics are available in CloudWatch automatic dashboards. For a full list of Amazon ECS Container Insights metrics, see Amazon ECS Container Insights Metrics in the *Amazon CloudWatch User Guide*.

Operational data is collected as performance log events. These are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates higher-level aggregated metrics at the cluster and service level as CloudWatch metrics. For more information, see Container Insights Structured Logs for Amazon ECS in the *Amazon CloudWatch User Guide*.

> **Important**
> Metrics collected by CloudWatch Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see CloudWatch Pricing. Amazon ECS also provides monitoring metrics that are provided at no additional cost. For more information, see Amazon ECS CloudWatch Metrics (p. 416).

Container Insights Considerations

The following should be considered when using CloudWatch Container Insights.

- CloudWatch Container Insights metrics only reflect the resources with running tasks during the specified time range. For example, if you have a cluster with one service in it but that service has no tasks in a `RUNNING` state, there will be no metrics sent to CloudWatch. If you have two services and one of them has running tasks and the other doesn't, only the metrics for the service with running tasks will be sent.
- Network metrics are only available for tasks that either use the Fargate launch type or use the `bridge` network mode.

Working with Container Insights-enabled clusters

Container Insights can be enabled for all new clusters created by opting in to the `containerInsights` account setting, on individual clusters by enabling it using the cluster settings during cluster creation, or on existing clusters by using the UpdateClusterSettings API.

Opting in to the `containerInsights` account setting can be done with both the Amazon ECS console and the AWS CLI. You must be running version `1.16.200` or later of the AWS CLI to use this feature. For more information on creating Amazon ECS clusters, see Creating a Cluster (p. 38).

> **Important**
> For clusters containing tasks or services using the EC2 launch type, your container instances must be running version 1.29.0 or later of the Amazon ECS agent. For more information, see Amazon ECS Container Agent Versions (p. 259).

To opt in all IAM users or roles on your account to Container Insights-enabled clusters using the console

1. As the root user of the account, 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 opt in to Container Insights-enabled clusters.
3. From the dashboard, choose **Account Settings**.
4. For **IAM user or role**, ensure your root user or container instance IAM role is selected.
5. For **Container Insights**, select the check box. Choose **Save** once finished.

   > **Important**
   > IAM users and IAM roles need the `ecs:PutAccountSetting` permission to perform this action.
6. On the confirmation screen, choose **Confirm** to save the selection.

To opt in all IAM users or roles on your account to Container Insights-enabled clusters using the command line

Any user on an account can 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)

```
aws ecs put-account-setting-default --name containerInsights --value enabled --region us-east-1
```

- Write-ECSAccountSettingDefault (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSettingDefault -Name containerInsights -Value enabled -Region us-east-1 -Force
```

To opt in an IAM user or container instance IAM role to Container Insights-enabled clusters as the root user using the command line

The root user on an account can use one of the following commands and specify the ARN of the principal IAM user or container instance IAM role in the request to modify the account settings.

- put-account-setting (AWS CLI)

The following example is for modifying the account setting of a specific IAM user:

```
aws ecs put-account-setting --name containerInsights --value enabled --principal-arn
 arn:aws:iam::aws_account_id:user/userName --region us-east-1
```

- Write-ECSAccountSetting (AWS Tools for Windows PowerShell)

  The following example is for modifying the account setting of a specific IAM user:

```
Write-ECSAccountSetting -Name containerInsights -Value enabled -PrincipalArn
 arn:aws:iam::aws_account_id:user/userName -Region us-east-1 -Force
```

To update the settings for an existing cluster using the command line

Use one of the following commands to update the setting for a cluster.

- update-cluster-settings (AWS CLI)

```
aws ecs update-cluster-settings --cluster cluster_name_or_arn --settings
 name=containerInsights,value=enabled|disabled --region us-east-1
```

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:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

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:

```
{
    "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"
}
```

Security in Amazon Elastic Container Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon Elastic Container Service, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon ECS. The following topics show you how to configure Amazon ECS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon ECS resources.

Topics

- Identity and Access Management for Amazon Elastic Container Service (p. 444)
- Logging and Monitoring in Amazon Elastic Container Service (p. 501)
- Compliance Validation for Amazon Elastic Container Service (p. 502)
- Infrastructure Security in Amazon Elastic Container Service (p. 503)

Identity and Access Management for Amazon Elastic Container Service

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon ECS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience (p. 445)
- Authenticating With Identities (p. 445)
- Managing Access Using Policies (p. 447)
- How Amazon Elastic Container Service Works with IAM (p. 448)
- Amazon Elastic Container Service Identity-Based Policy Examples (p. 452)
- Supported Resource-Level Permissions for Amazon ECS API Actions (p. 464)
- Managed Policies and Trust Relationships (p. 465)
- Service-Linked Role for Amazon ECS (p. 473)

- Amazon ECS Task Execution IAM Role (p. 482)
- Amazon ECS Container Instance IAM Role (p. 486)
- IAM Roles for Tasks (p. 489)
- Amazon ECS CodeDeploy IAM Role (p. 493)
- Amazon ECS CloudWatch Events IAM Role (p. 496)
- Troubleshooting Amazon Elastic Container Service Identity and Access (p. 499)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon ECS.

Service user – If you use the Amazon ECS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon ECS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon ECS, see Troubleshooting Amazon Elastic Container Service Identity and Access (p. 499).

Service administrator – If you're in charge of Amazon ECS resources at your company, you probably have full access to Amazon ECS. It's your job to determine which Amazon ECS features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon ECS, see How Amazon Elastic Container Service Works with IAM (p. 448).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon ECS. To view example Amazon ECS identity-based policies that you can use in IAM, see Amazon Elastic Container Service Identity-Based Policy Examples (p. 452).

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and

is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access Control Lists (ACLs)

Access control lists (ACLs) are a type of policy that controls which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they

do not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) Overview in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the *IAM User Guide*.

How Amazon Elastic Container Service Works with IAM

Before you use IAM to manage access to Amazon ECS, you should understand what IAM features are available to use with Amazon ECS. To get a high-level view of how Amazon ECS and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

- Amazon ECS Identity-Based Policies (p. 448)
- Amazon ECS Resource-Based Policies (p. 452)
- Authorization Based on Amazon ECS Tags (p. 452)
- Amazon ECS IAM Roles (p. 452)

Amazon ECS Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon ECS supports specific actions, resources,

and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon ECS use the following prefix before the action: `ecs:`. For example, to grant someone permission to create an Amazon ECS cluster with the Amazon ECS `CreateCluster` API operation, you include the `ecs:CreateCluster` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon ECS defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "ecs:action1",
      "ecs:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "ecs:Describe*"
```

To see a list of Amazon ECS actions, see Actions, Resources, and Condition Keys for Amazon Elastic Container Service in the *IAM User Guide*

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The Amazon ECS cluster resource has the following ARN:

```
arn:${Partition}:ecs:${Region}:${Account}:cluster/${clusterName}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `my-cluster` cluster in your statement, use the following ARN:

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster"
```

To specify all clusters that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

Some Amazon ECS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Some Amazon ECS API actions can be performed on multiple resources. For example, multiple clusters can be referenced when calling the `DescribeClusters` API action. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
```

The following table describes the ARNs for each resource type used by the Amazon ECS API actions.

Important
The following table uses the new longer ARN format for Amazon ECS tasks, services, and container instances. If you have not opted in to the long ARN format, the ARNs will not include the cluster name. For more information, see Amazon Resource Names (ARNs) and IDs (p. 184).

Resource Type	ARN
All Amazon ECS resources	arn:aws:ecs:*
All Amazon ECS resources owned by the specified account in the specified region	arn:aws:ecs:*region*:*account*:*
Cluster	arn:aws:ecs:*region*:*account*:cluster/*cluster-name*
Container instance	arn:aws:ecs:*region*:*account*:container-instance/*cluster-name*/*container-instance-id*
Task definition	arn:aws:ecs:*region*:*account*:task-definition/*task-definition-family-name*:*task-definition-revision-number*
Service	arn:aws:ecs:*region*:*account*:service/*cluster-name*/*service-name*
Task	arn:aws:ecs:*region*:*account*:task/*cluster-name*/*task-id*
Container	arn:aws:ecs:*region*:*account*:container/*container-id*

To learn with which actions you can specify the ARN of each resource, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 464).

Condition Keys

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the *IAM User Guide*.

Amazon ECS defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Amazon ECS implements the following service-specific condition keys.

Condition Key	Description	Evaluation Types
aws:RequestTag/${TagKey}	The context key is formatted `"aws:RequestTag/tag-key":"tag-value"` where *tag-key* and *tag-value* are a tag key and value pair. Checks that the tag key–value pair is present in an AWS request. For example, you could check to see that the request includes the tag key `"Dept"` and that it has the value `"Accounting"`.	String
aws:ResourceTag/${TagKey}	The context key is formatted `"aws:ResourceTag/tag-key":"tag-value"` where *tag-key* and *tag-value* are a tag key and value pair. Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.	String
aws:TagKeys	This context key is formatted `"aws:TagKeys":"tag-key"` where *tag-key* is a list of tag keys without values (for example, `["Dept","Cost-Center"]`). Checks the tag keys that are present in an AWS request.	String
ecs:ResourceTag/${TagKey}	The context key is formatted `"ecs:ResourceTag/tag-key":"tag-value"` where *tag-key* and *tag-value* are a tag key and value pair. Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.	String
ecs:cluster	The context key is formatted `"ecs:cluster":"cluster-arn"` where *cluster-arn* is the ARN for the Amazon ECS cluster.	ARN, Null
ecs:container-instances	The context key is formatted `"ecs:container-instances":"container-instance-arns"` where *container-instance-arns* is one or more container instance ARNs.	ARN, Null
ecs:task-definition	The context key is formatted `"ecs:task-definition":"task-definition-arn"` where *task-definition-arn* is the ARN for the Amazon ECS task definition.	ARN, Null
ecs:service	The context key is formatted `"ecs:service":"service-arn"` where *service-arn* is the ARN for the Amazon ECS service.	ARN, Null

To learn with which actions and resources you can use a condition key, see Supported Resource-Level Permissions for Amazon ECS API Actions (p. 464).

Examples

To view examples of Amazon ECS identity-based policies, see Amazon Elastic Container Service Identity-Based Policy Examples (p. 452).

Amazon ECS Resource-Based Policies

Amazon ECS does not support resource-based policies.

Authorization Based on Amazon ECS Tags

You can attach tags to Amazon ECS resources or pass tags in a request to Amazon ECS. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:RequestTag/key-name` or `aws:TagKeys` condition keys. For more information, see Controlling Access Using Tags in the *IAM User Guide*.

For more information about tagging Amazon ECS resources, see Resources and Tags (p. 407).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Describing Amazon ECS Services Based on Tags (p. 463).

Amazon ECS IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon ECS

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon ECS supports using temporary credentials.

Service-Linked Roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon ECS supports service-linked roles. For details about creating or managing Amazon ECS service-linked roles, see Service-Linked Role for Amazon ECS (p. 473).

Service Roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon ECS supports service roles.

Amazon Elastic Container Service Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon ECS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

- Policy Best Practices (p. 453)
- Allow Users to View Their Own Permissions (p. 453)
- Amazon ECS First Run Wizard Permissions (p. 454)
- Cluster Examples (p. 458)
- Container Instance Examples (p. 459)
- Task Definition Examples (p. 460)
- Run Task Example (p. 460)
- Start Task Example (p. 461)
- List and Describe Task Examples (p. 461)
- Create Service Example (p. 462)
- Update Service Example (p. 463)
- Describing Amazon ECS Services Based on Tags (p. 463)

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon ECS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon ECS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                  "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS First Run Wizard Permissions

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. 465) managed policy below shows the required permissions to complete the Amazon ECS first-run wizard.

```
{
    "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",
                "appmesh:ListMeshes",
                "appmesh:ListVirtualNodes",
                "appmesh:DescribeVirtualNode",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:DeleteAutoScalingGroup",
                "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: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",
                "servicediscovery:GetOperation",
                "servicediscovery:GetService",
                "servicediscovery:ListNamespaces",
                "servicediscovery:ListServices",
                "servicediscovery:UpdateService",
                "servicediscovery:DeleteService"
            ],
            "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"
                    ]
                }
            }
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/ecsAutoscaleRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "application-autoscaling.amazonaws.com",
                        "application-autoscaling.amazonaws.com.cn"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": [
                        "ecs.amazonaws.com",
                        "spot.amazonaws.com",
                        "spotfleet.amazonaws.com",
                        "ecs.application-autoscaling.amazonaws.com",
                        "autoscaling.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

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:
  - Service Scheduler IAM Role (p. 479)
  - Amazon ECS Container Instance IAM Role (p. 486)
  - Amazon ECS Task Execution IAM Role (p. 482)

Cluster Examples

The following IAM policy allows permission to create and list clusters. The `CreateCluster` and `ListClusters` actions do not accept any resources, so the resource definition is set to `*` for all resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:ListClusters"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The following IAM policy allows permission to describe and delete a specific cluster. The `DescribeClusters` and `DeleteCluster` actions accept cluster ARNs as resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeClusters",
        "ecs:DeleteCluster"
      ],
      "Resource": [
        "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/<cluster_name>"
      ]
    }
  ]
}
```

The following IAM policy can be attached to a user or group that would only allow that user or group to perform operations on a specific cluster.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecs:Describe*",
        "ecs:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "ecs:DeleteCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:ListContainerInstances",
        "ecs:RegisterContainerInstance",
        "ecs:SubmitContainerStateChange",
```

```
        "ecs:SubmitTaskStateChange"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
    },
    {
      "Action": [
        "ecs:DescribeContainerInstances",
        "ecs:DescribeTasks",
        "ecs:ListTasks",
        "ecs:UpdateContainerAgent",
        "ecs:StartTask",
        "ecs:StopTask",
        "ecs:RunTask"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
        }
      }
    }
  ]
}
```

Container Instance Examples

Container instance registration is handled by the Amazon ECS agent, but there may be times where you want to allow a user to deregister an instance manually from a cluster. Perhaps the container instance was accidentally registered to the wrong cluster, or the instance was terminated with tasks still running on it.

The following IAM policy allows a user to list and deregister container instances in a specified cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DeregisterContainerInstance",
        "ecs:ListContainerInstances"
      ],
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
      ]
    }
  ]
}
```

The following IAM policy allows a user to describe a specified container instance in a specified cluster. To open this permission up to all container instances in a cluster, you can replace the container instance UUID with *.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeContainerInstances"
```

```
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:container-instance/
<container_instance_UUID>"
      ]
    }
  ]
}
```

Task Definition Examples

Task definition IAM policies do not support resource-level permissions, but the following IAM policy allows a user to register, list, and describe task definitions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:RegisterTaskDefinition",
        "ecs:ListTaskDefinitions",
        "ecs:DescribeTaskDefinition"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Run Task Example

The resources for `RunTask` are task definitions. To limit which clusters a user can run task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to run any revision of a specific task definition on a specific cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:RunTask"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task-definition/<task_family>:*"
```

```
      ]
    }
  ]
}
```

Start Task Example

The resources for `StartTask` are task definitions. To limit which clusters and container instances a user can start task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to start any revision of a specific task definition on a specific cluster and specific container instance.

Note
For this example, when you call the `StartTask` API with the AWS CLI or another AWS SDK, you must specify the task definition revision so that the `Resource` mapping matches.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:StartTask"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>",
          "ecs:container-instances" : [
            "arn:aws:ecs:<region>:<aws_account_id>:container-instance/
<container_instance_UUID>"
          ]
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task-definition/<task_family>:*"
      ]
    }
  ]
}
```

List and Describe Task Examples

The following IAM policy allows a user to list tasks for a specified cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:ListTasks"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
        "*"
```

```
      ]
    }
  ]
}
```

The following IAM policy allows a user to describe a specified task in a specified cluster:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeTasks"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task/<task_UUID>"
      ]
    }
  ]
}
```

Create Service Example

The following IAM policy allows a user to create Amazon ECS services in the AWS Management Console:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:Describe*",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "ecs:List*",
        "ecs:Describe*",
        "ecs:CreateService",
        "elasticloadbalancing:Describe*",
        "iam:AttachRolePolicy",
        "iam:CreateRole",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:ListAttachedRolePolicies",
        "iam:ListRoles",
        "iam:ListGroups",
        "iam:ListUsers"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Update Service Example

The following IAM policy allows a user to update Amazon ECS services in the AWS Management Console:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:Describe*",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "ecs:List*",
        "ecs:Describe*",
        "ecs:UpdateService",
        "iam:AttachRolePolicy",
        "iam:CreateRole",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
        "iam:GetRole",
        "iam:ListAttachedRolePolicies",
        "iam:ListRoles",
        "iam:ListGroups",
        "iam:ListUsers"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Describing Amazon ECS Services Based on Tags

You can use conditions in your identity-based policy to control access to Amazon ECS resources based on tags. This example shows how you might create a policy that allows describing your services. However, permission is granted only if the service tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeServices",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "*"
        },
        {
            "Sid": "ViewServiceIfOwner",
            "Effect": "Allow",
            "Action": "ecs:DescribeServices",
            "Resource": "arn:aws:ecs:*:*:service/*",
            "Condition": {
                "StringEquals": {"ecs:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
```

```
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to describe an Amazon ECS service, the service must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

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.

For more information about the resources that are created or modified by the Amazon ECS actions, and the ARNs and Amazon ECS condition keys that you can use in an IAM policy statement, see Actions, Resources, and Condition Keys for Amazon Elastic Container Service in the *IAM User Guide*.

Considerations for Resource-Level Permissions

When controlling access to Amazon ECS API actions by specifying the Amazon Resource Name (ARN) of a resource in an IAM policy, be mindful that ECS has introduced an account setting that affects the ARN format for container instances, services, and tasks. To use resource-level permissions, we recommend that you opt-in to the new, longer ARN format. For more information, see Amazon Resource Names (ARNs) and IDs (p. 184).

When an IAM policy is evaluated, the specified resources are evaluated based on their use of the new, longer ARN format. The following are examples of how access is controlled.

Specifying a Service with a Cluster Only with a Wildcard

Example: `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`cluster_name`*`*`

In this example, access will be controlled to the following services:

- All services using the new ARN format that are in the `cluster_name*` cluster.
- All services using the old ARN format that are in the `cluster_name*` cluster.

> **Important**
> This will **NOT** control access to services using the old ARN format that have a service name with the `cluster_name` prefix that are not in the `cluster_name*` cluster.

Specifying a Service with both a Cluster and Service Name with a Wildcard

Example: `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`cluster_name`*`/`*`service_name`*`*`

In this example, access will be controlled to the following services:

- All services using the new ARN format that are in the `cluster_name` cluster with the `service_name` prefix.
- All services using the old ARN format that are in the `cluster_name` cluster with the `service_name` prefix, even though the actual ARN of the service will still have the `arn:aws:ecs:`*`region`*`:`*`aws_account_id`*`:service/`*`service_name*` ARN format.

Specifying a Service with a full ARN

Example: `arn:aws:ecs:region:aws_account_id:service/cluster_name/service_name`

In this example, access will be controlled to the following services:

- All services using the new ARN format that are in the `cluster_name` cluster with the `service_name` service name.
- All services using the old ARN format that are in the `cluster_name` cluster with the `service_name` service name, even though the actual ARN of the service will still have the `arn:aws:ecs:region:aws_account_id:service/service_name` ARN format.

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. 465)
- Amazon ECR Managed Policies (p. 471)

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. 465)
- AmazonEC2ContainerServiceFullAccess (p. 469)
- AmazonEC2ContainerServiceforEC2Role (p. 469)
- AmazonEC2ContainerServiceRole (p. 470)
- AmazonEC2ContainerServiceAutoscaleRole (p. 470)
- AmazonEC2ContainerServiceTaskRole (p. 471)
- AmazonEC2ContainerServiceEventsRole (p. 471)

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.

```
{
    "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",
                "appmesh:ListMeshes",
                "appmesh:ListVirtualNodes",
                "appmesh:DescribeVirtualNode",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:CreateLaunchConfiguration",
                "autoscaling:DeleteAutoScalingGroup",
                "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: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",
                "servicediscovery:GetOperation",
                "servicediscovery:GetService",
                "servicediscovery:ListNamespaces",
                "servicediscovery:ListServices",
                "servicediscovery:UpdateService",
                "servicediscovery:DeleteService"
            ],
            "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"
                    ]
                }
            }
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/ecsAutoscaleRole*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "application-autoscaling.amazonaws.com",
                        "application-autoscaling.amazonaws.com.cn"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": [
                        "ecs.amazonaws.com",
                        "spot.amazonaws.com",
                        "spotfleet.amazonaws.com",
                        "ecs.application-autoscaling.amazonaws.com",
                        "autoscaling.amazonaws.com"
                    ]
                }
            }
        }
```

```
    ]
}
```

AmazonEC2ContainerServiceFullAccess

This managed policy allows full administrator access to Amazon ECS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:Describe*",
                "autoscaling:UpdateAutoScalingGroup",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStack*",
                "cloudformation:UpdateStack",
                "cloudwatch:GetMetricStatistics",
                "ec2:Describe*",
                "elasticloadbalancing:*",
                "ecs:*",
                "events:DescribeRule",
                "events:DeleteRule",
                "events:ListRuleNamesByTarget",
                "events:ListTargetsByRule",
                "events:PutRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "iam:ListInstanceProfiles",
                "iam:ListRoles",
                "iam:PassRole"
            ],
            "Resource": "*"
        }
    ]
}
```

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. 486).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags",
                "ecs:CreateCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:DiscoverPollEndpoint",
                "ecs:Poll",
                "ecs:RegisterContainerInstance",
                "ecs:StartTelemetrySession",
                "ecs:UpdateContainerInstancesState",
                "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 Service Scheduler IAM Role (p. 479).

```
{
    "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 Service Auto Scaling IAM Role (p. 480).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm"
            ],
            "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 IAM Roles for Tasks (p. 489).

```
{
  "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. 330).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        }
    ]
}
```

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. 472)
- AmazonEC2ContainerRegistryPowerUser (p. 472)
- AmazonEC2ContainerRegistryReadOnly (p. 473)

AmazonEC2ContainerRegistryFullAccess

This managed policy is a starting point for customers who are looking to provide an IAM user or role with full administrator access to manage their use of Amazon ECR. The Amazon ECR Lifecycle Policies feature enables customers to specify the lifecycle management of images in a repository. Lifecycle policy events are reported as CloudTrail events, and Amazon ECR is integrated with AWS CloudTrail to display a customer's lifecycle policy events directly in the Amazon ECR console. The `AmazonEC2ContainerRegistryFullAccess` managed IAM policy includes the `cloudtrail:LookupEvents` permission to facilitate this behavior.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "cloudtrail:LookupEvents"
            ],
            "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:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

AmazonEC2ContainerRegistryReadOnly

This managed policy allows read-only access to Amazon ECR, such as the ability to list repositories and the images within the repositories, and also to pull images from Amazon ECR with the Docker CLI.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings"
            ],
            "Resource": "*"
        }
    ]
}
```

Service-Linked Role for Amazon ECS

Amazon Elastic Container Service uses a service-linked role for the permissions it requires to call other AWS services on your behalf. For more information, see Using Service-Linked Roles in the *IAM User Guide*.

Prior to the introduction of a service-linked role for Amazon ECS, you were required to create an IAM role for your Amazon ECS services which granted Amazon ECS the permission it needed. This role is no longer required, however it is available if needed. For more information, see Legacy IAM Roles for Amazon ECS (p. 479).

Permissions Granted by the Service-Linked Role

Amazon ECS uses the service-linked role named **AWSServiceRoleForECS** to enable Amazon ECS to call AWS APIs on your behalf.

The **AWSServiceRoleForECS** service-linked role trusts the `ecs.amazonaws.com` service principal to assume the role.

The role permissions policy allows Amazon ECS to complete the following actions on resources.

```
{
    "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": "AutoScaling",
        "Effect": "Allow",
        "Action": [
            "autoscaling:Describe*"
        ],
        "Resource": "*"
    },
    {
        "Sid": "AutoScalingManagement",
        "Effect": "Allow",
        "Action": [
            "autoscaling:DeletePolicy",
            "autoscaling:PutScalingPolicy",
            "autoscaling:SetInstanceProtection",
            "autoscaling:UpdateAutoScalingGroup"
        ],
        "Resource": "*",
        "Condition": {
            "Null": {
                "autoscaling:ResourceTag/AmazonECSManaged": "false"
            }
        }
    },
    {
        "Sid": "AutoScalingPlanManagement",
        "Effect": "Allow",
        "Action": [
            "autoscaling-plans:CreateScalingPlan",
            "autoscaling-plans:DeleteScalingPlan",
            "autoscaling-plans:DescribeScalingPlans"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CWAlarmManagement",
        "Effect": "Allow",
        "Action": [
            "cloudwatch:DeleteAlarms",
            "cloudwatch:DescribeAlarms",
            "cloudwatch:PutMetricAlarm"
        ],
```

```
            "Resource": "arn:aws:cloudwatch:*:*:alarm:*"
        },
        {
            "Sid": "ECSTagging",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*"
        },
        {
            "Sid": "CWLogGroupManagement",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DescribeLogGroups",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/ecs/*"
        },
        {
            "Sid": "CWLogStreamManagement",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/ecs/*:log-stream:*"
        }
    ]
}
```

Create the Service-Linked Role

Under most circumstances, you don't need to manually create the service-linked role. For example, when you create a new cluster (for example, with the Amazon ECS first-run experience, 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.

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:

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ecs.amazonaws.com/
AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}
```

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
```

Edit the Service-Linked Role

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*.

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:

```
{
    "Effect": "Allow",
    "Action": [
        "iam:UpdateRoleDescription"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ecs.amazonaws.com/
AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}
```

Delete the Service-Linked Role

If you no longer use Amazon ECS, we recommend that you delete the service-linked 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.

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:

```
{
    "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"}}
}
```

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. 363) and Deleting a Service (p. 366).
2. Force deregister all container instances from all clusters in all regions. For more information, see Deregister a Container Instance (p. 248).
3. Delete all Amazon ECS clusters in all regions. For more information, see Deleting a Cluster (p. 71).

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. 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. Enter the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name AWSServiceRoleForECS+OPTIONAL-SUFFIX
```

2. Use 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.

Legacy IAM Roles for Amazon ECS

Prior to the introduction of the **AWSServiceRoleForECS** IAM role, you were required to create separate IAM roles to enable Amazon ECS permissions to call Elastic Load Balancing and Application Auto Scaling APIs on your behalf.

The Amazon ECS service scheduler IAM role grants the Amazon ECS service scheduler permissions that it needs to register and deregister container instances with your load balancers. You can optionally create the service scheduler IAM role and specify it when creating a service, or preferably you can allow Amazon ECS to use the service-linked role.

The Amazon ECS Service Auto Scaling IAM role grants Amazon ECS permission to describe your CloudWatch alarms and registered services, as well as permission to update your Amazon ECS service's desired count on your behalf.

These legacy IAM roles are described in more detail below, but have effectively been replaced by the Amazon ECS service-linked role.

Service Scheduler IAM Role

Amazon ECS provides a managed IAM policy named `AmazonEC2ContainerServiceRole` to use for the service scheduler IAM role. The `AmazonEC2ContainerServiceRole` policy is shown below.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:Describe*",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:Describe*",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets"
            ],
            "Resource": "*"
        }
    ]
}
```

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**.
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": "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. For **Select type of trusted entity** section, choose **AWS service**.
4. For **Choose the service that will use this role**, choose **Elastic Container Service**.
5. For **Select your use case**, choose **Elastic Container Service** and choose **Next: Permissions**.
6. In the **Attached permissions policy** section, select the **AmazonEC2ContainerServiceRole** policy and choose **Next: Review**.
7. For **Role Name**, type `ecsServiceRole`, enter a **Role description** and then choose **Create role**.

Service Auto Scaling IAM Role

Amazon ECS provides a managed IAM policy named `AmazonEC2ContainerServiceAutoscaleRole` to use for the Service Auto Scaling IAM role. The `AmazonEC2ContainerServiceAutoscaleRole` policy is shown below.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms",
                "cloudwatch:PutMetricAlarm"
            ],
            "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 **Permissions 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 policies**.
   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**.
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": [
    {
      "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 role**.
3. In the **Choose the service that will use this role** section, choose **Elastic Container Service**.
4. In the **Select your use case** section, choose **Elastic Container Service Autoscale**, **Next: Permissions**.
5. For **Add tags (optional)**, enter any key value tags you wish to add to the IAM role. Choose **Next: Review** when finished.
6. In the **Role name** field, type `ecsAutoscaleRole` to name the role, and then choose **Create Role** to finish.

Amazon ECS Task Execution IAM Role

The Amazon ECS container agent, and the Fargate agent for your Fargate tasks, make calls to the Amazon ECS API on your behalf. The agent requires an IAM role for the service to know that the agent belongs to you. This IAM role is referred to as a task execution IAM role. You can have multiple task execution roles for different purposes associated with your account.

The following are common use cases for a task execution IAM role:

- Your task uses the Fargate launch type and...
  - is pulling a container image from Amazon ECR.
  - uses the `awslogs` or `awsfirelens` log driver. For more information, see Using the awslogs Log Driver (p. 143) and Custom Log Routing (p. 150).
- 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. 484).
  - 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. 484).

  Note
  The task execution role is supported by Amazon ECS container agent version 1.16.0 and later.

Amazon ECS provides the managed policy named `AmazonECSTaskExecutionRolePolicy` which contains the permissions the common use cases described above require. It may be necessary to add inline policies to your task execution role for special use cases which are outlined below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

An 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, see Creating the task execution IAM role (p. 483). If the role does exist, select the role to view the attached policies.
4. On the **Permissions** tab, 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 policies**.
   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**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Creating the task execution IAM role

If your account does not already have a task execution role, use the following steps to create the role.

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. 160).

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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:secret_name",
        "arn:aws:kms:<region>:<aws_account_id>:key/key_id"
      ]
    }
  ]
}
```

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. 163).

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:

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:<secret_name>",
        "arn:aws:kms:<region>:<aws_account_id>:key/<key_id>"
      ]
    }
  ]
}
```

Optional IAM Permissions for Fargate Tasks Pulling Amazon ECR Images over Interface Endpoints

When launching tasks that use the Fargate launch type that pull images from Amazon ECR when Amazon ECR is configured to use an interface VPC endpoint, you can restrict the tasks access to a specific VPC or VPC endpoint. Do this by creating a task execution role for the tasks to use that use IAM condition keys.

Use the following IAM global condition keys to restrict access to a specific VPC or VPC endpoint. For more information, see AWS Global Condition Context Keys.

- `aws:SourceVpc`—Restricts access to a specific VPC.
- `aws:SourceVpce`—Restricts access to a specific VPC endpoint.

The following task execution role policy provides an example for adding condition keys:

Important
The `ecr:GetAuthorizationToken` API action cannot have the `aws:sourceVpc` or `aws:sourceVpce` condition keys applied to it because the GetAuthorizationToken API call goes through the elastic network interface owned by AWS Fargate rather than the elastic network interface of the task.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpce": "vpce-xxxxxx",
```

```
                    "aws:sourceVpc": "vpc-xxxxx"
                }
            }
        }
    ]
}
```

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. This IAM 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. 489).
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. 489)) 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. 85).

```
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": [
```

```
                "ec2:DescribeTags",
                "ecs:CreateCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:DiscoverPollEndpoint",
                "ecs:Poll",
                "ecs:RegisterContainerInstance",
                "ecs:StartTelemetrySession",
                "ecs:UpdateContainerInstancesState",
                "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 in the next section 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.

   Important
   The **AmazonEC2ContainerServiceforEC2Role** managed policy should be attached to the container instance IAM role, otherwise you will receive an error using the AWS Management Console to create clusters.

   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**.

```
{
  "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**.

   Important
   The **AmazonEC2ContainerServiceforEC2Role** managed policy should be attached to the container instance IAM role, otherwise you will receive an error using the AWS Management Console to create clusters.
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. 283).

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**.
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. 486).
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.

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. 486).
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. 270).
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. 85).

```
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
  ```

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:

```
curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

Output:

```
{
    "AccessKeyId": "ACCESS_KEY_ID",
    "Expiration": "EXPIRATION_DATE",
    "RoleArn": "TASK_ROLE_ARN",
    "SecretAccessKey": "SECRET_ACCESS_KEY",
    "Token": "SECURITY_TOKEN_STRING"
}
```

Note
The default expiration time for the generated IAM role credentials is 6 hours.

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. 491) and Using a Supported AWS SDK (p. 493).

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. 703).

Topics

- Benefits of Using IAM Roles for Tasks (p. 490)
- Enabling Task IAM Roles on your Container Instances (p. 491)
- Creating an IAM Role and Policy for your Tasks (p. 491)
- Using a Supported AWS SDK (p. 493)
- Specifying an IAM Role for your Tasks (p. 493)

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. 264). 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. 190).

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. 270)):

`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. 268). You will also need to set the following networking commands on your container instance so that the containers in your tasks can retrieve their AWS credentials:

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --to-
ports 51679
```

You must save these **iptables** rules on your container instance for them to survive a reboot. You can use the **iptables-save** and **iptables-restore** commands to save your **iptables** rules and restore them at boot. For more information, consult your specific operating system documentation.

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.

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.

The Amazon ECS Task Role trust relationship is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

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 Amazon Resource Name (ARN) of your Amazon S3 bucket, and then choose **Review policy**.
4. On the **Review policy** page, for **Name** type your own unique name, such as `AmazonECSTaskS3BucketPolicy`.
5. 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.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:GetObject"
         ],
         "Resource": [
           "arn:aws:s3:::my-task-secrets-bucket/*"
         ]
       }
     ]
   }
   ```

2. Choose **Create policy**.

To create an IAM role for your tasks

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 **Elastic Container Service**.
5. For **Select your use case**, choose **Elastic Container Service Task** and choose **Next: Permissions**.
6. For **Attach permissions policy**, select the policy to use for your tasks (in this example `AmazonECSTaskS3BucketPolicy`, and then choose **Next: Tags**.
7. For **Add tags (optional)**, enter any metadata tags you want to associate with the IAM role, and then choose **Next: Review**.
8. 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 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. 75).

  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. 316).

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 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. 370).

There are two managed policies provided. The `AWSCodeDeployRoleForECS` policy, shown below, gives CodeDeploy permission to update any resource using the associated action.

```
{
    "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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:DescribeServices",
                "ecs:CreateTaskSet",
                "ecs:UpdateServicePrimaryTaskSet",
                "ecs:DeleteTaskSet",
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:CodeDeployTopic_*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:ModifyRule"
            ],
            "Resource": "*",
            "Effect": "Allow"
```

```
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:*:*:function:CodeDeployHook_*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectMetadata",
                "s3:GetObjectVersion"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "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 - ECS**, **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**.
4. Choose **Trust Relationships**, **Edit trust relationship**.
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. 482) and IAM Roles for Tasks (p. 489).

   Follow the substeps below to create an inline policy.

   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   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 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. 330).

The `AmazonEC2ContainerServiceEventsRole` policy is shown below.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        }
    ]
}
```

If your scheduled tasks require the use of the task execution role, a task role, or a task role override, then you must add `iam:PassRole` permissions for each task execution role, task role, or task role override to the CloudWatch Events IAM role. For more information about the task execution role, see Amazon ECS Task Execution IAM Role (p. 482).

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 **Permissions 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 policies**.
   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: Tags**.
5. In the **Add tags (optional)** section, enter any tags you would like to associate with the role and choose **Next: Review**.
6. For **Role name**, type `ecsEventsRole` to name the role, optionally enter a description, and then choose **Create role**.
7. Review your role information and choose **Create Role**.
8. Search the list of roles for `ecsEventsRole` and select the role you just created.
9. Choose **Trust relationships**, **Edit trust relationship**.
10. Replace the existing trust relationship with the following text in 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 add permissions for the task execution role to the CloudWatch Events IAM role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, **Create policy**.
3. Choose **JSON**, paste the following policy, and then choose **Review policy**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::<aws_account_id>:role/
<ecsTaskExecutionRole_or_TaskRole_name>"
            ]
        }
    ]
}
```

4. For **Name**, type `AmazonECSEventsTaskExecutionRole`, optionally enter a description, and then choose **Create policy**.
5. In the navigation pane, choose **Roles**.
6. Search the list of roles for `ecsEventsRole` and select the role to view the attached policies.
7. Choose **Attach policy**.
8. In the **Attach policy** section, select the **AmazonECSEventsTaskExecutionRole** policy and choose **Attach policy**.

Troubleshooting Amazon Elastic Container Service Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon ECS and IAM.

Topics

- I Am Not Authorized to Perform an Action in Amazon ECS (p. 499)
- I Am Not Authorized to Perform iam:PassRole (p. 500)
- I Want to View My Access Keys (p. 500)
- I'm an Administrator and Want to Allow Others to Access Amazon ECS (p. 500)
- I Want to Allow People Outside of My AWS Account to Access My Amazon ECS Resources (p. 501)

I Am Not Authorized to Perform an Action in Amazon ECS

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a *`widget`* but does not have `ecs:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 ecs:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-widget`* resource using the `ecs:`*`GetWidget`* action.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon ECS.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon ECS. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the *IAM User Guide*.

I'm an Administrator and Want to Allow Others to Access Amazon ECS

To allow others to access Amazon ECS, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon ECS.

To get started right away, see Creating Your First IAM Delegated User and Group in the *IAM User Guide*.

I Want to Allow People Outside of My AWS Account to Access My Amazon ECS Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon ECS supports these features, see How Amazon Elastic Container Service Works with IAM (p. 448).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Logging and Monitoring in Amazon Elastic Container Service

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Elastic Container Service and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon ECS resources and responding to potential incidents:

Amazon CloudWatch Alarms

Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Amazon ECS CloudWatch Metrics (p. 416).

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. 424).

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. 143).

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. 237).

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 Events and EventBridge (p. 428) in this guide and What Is Amazon CloudWatch Events? in the *Amazon CloudWatch Events User Guide*.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon ECS. Using the information collected by CloudTrail, you can determine the request that was made to Amazon ECS, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Logging Amazon ECS API Calls with AWS CloudTrail (p. 441).

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

For more information, see AWS Trusted Advisor in the *AWS Support User Guide*.

Another important part of monitoring Amazon ECS involves manually monitoring those items that the CloudWatch alarms don't cover. The CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your container instances and the containers in your tasks.

Compliance Validation for Amazon Elastic Container Service

Third-party auditors assess the security and compliance of Amazon Elastic Container Service as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon ECS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Infrastructure Security in Amazon Elastic Container Service

As a managed service, Amazon Elastic Container Service is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon ECS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location, but Amazon ECS does support resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon ECS policies to control access from specific Amazon Virtual Private Cloud endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon ECS resource from only the specific VPC within the AWS network. For more information, see Amazon ECS interface VPC endpoints (AWS PrivateLink) (p. 503).

Topics

- Amazon ECS interface VPC endpoints (AWS PrivateLink) (p. 503)

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.

For more information about AWS PrivateLink and VPC endpoints, see VPC Endpoints in the *Amazon VPC User Guide*.

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, Secrets Manager, or Amazon CloudWatch Logs described in the following points.
  - To allow your tasks to pull private images from Amazon ECR, you must create the interface VPC endpoints for Amazon ECR. For more information, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.

    Important
    If you configure Amazon ECR to use an interface VPC endpoint, you can create a task execution role that includes condition keys to restrict access to a specific VPC or VPC

endpoint. For more information, see Optional IAM Permissions for Fargate Tasks Pulling Amazon ECR Images over Interface Endpoints (p. 485).

- To allow your tasks to pull sensitive data from Secrets Manager, you must create the interface VPC endpoints for Secrets Manager. For more information, see Using Secrets Manager with VPC Endpoints in the *AWS Secrets Manager User Guide*.
- If your VPC doesn't have an internet gateway and your tasks use the `awslogs` log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see Using CloudWatch Logs with Interface VPC Endpoints in the *Amazon CloudWatch Logs User Guide*.

- Tasks using the EC2 launch type require that the container instances that they're launched on to run version `1.25.1` or later of the Amazon ECS container agent. For more information, see Amazon ECS Container Agent Versions (p. 259).
- 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. 236).
2. Stop the container agent.

```
sudo docker stop ecs-agent
```

3. Start the container agent.

```
sudo docker start ecs-agent
```

After you have created the VPC endpoints and restarted the Amazon ECS container agent on each container instance, all newly launched tasks pick up the new configuration.

Create the Secrets Manager and Systems Manager endpoints

If you are referencing either Secrets Manager secrets or Systems Manager Parameter Store parameters in your task definitions to inject sensitive data into your containers, you need to create the interface VPC endpoints for Secrets Manager or Systems Manager so those tasks can reach those services. You only need to create the endpoints from the specific service your sensitive data is hosted in. For more information, see Specifying Sensitive Data (p. 163).

For more information about Secrets Manager VPC endpoints, see Using Secrets Manager with VPC endpoints in the *AWS Secrets Manager User Guide*.

For more information about Systems Manager VPC endpoints, see Using Systems Manager with VPC endpoints in the *AWS Systems Manager User Guide*.

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, a popular open-source specification for defining and running multi-container applications. Use the ECS CLI as part of your everyday development and testing cycle as an alternative to the AWS Management Console.

> **Important**
> At this time, the latest version of the Amazon ECS CLI only supports the major versions of Docker Compose file syntax versions 1, 2, and 3. The version specified in the compose file must be the string `"1"`, `"1.0"`, `"2"`, `"2.0"`, `"3"`, or `"3.0"`. Docker Compose minor versions are not supported.

The latest version of the Amazon ECS CLI is 1.17.0. 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. 506)
- Configuring the Amazon ECS CLI (p. 512)
- Migrating Configuration Files (p. 513)
- Tutorial: Creating a Cluster with a Fargate Task Using the Amazon ECS CLI (p. 514)
- Tutorial: Creating a Cluster with an EC2 Task Using the Amazon ECS CLI (p. 519)
- Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 522)
- Amazon ECS Command Line Reference (p. 525)

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:

```
sudo curl -o /usr/local/bin/ecs-cli https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-amd64-latest
```

- For Linux systems:

```
sudo curl -o /usr/local/bin/ecs-cli https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-linux-amd64-latest
```

- For Windows systems:

Open Windows PowerShell and run the following commands:

```
PS C:\> New-Item -Path 'C:\Program Files\Amazon\ECSCLI' -ItemType
 Directory
PS C:\> Invoke-WebRequest -OutFile 'C:\Program Files\Amazon\ECSCLI\ecs-cli.exe' https://
amazon-ecs-cli.s3.amazonaws.com/ecs-cli-windows-amd64-latest.exe
```

Note
If you encounter permissions issues, ensure that you are running PowerShell as an administrator.

Step 2: 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:\> Invoke-WebRequest -OutFile md5.txt https://
amazon-ecs-cli.s3.amazonaws.com/ecs-cli-windows-amd64-
latest.md5
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.
- For Windows systems, download and use the Windows simple installer from the GnuPG website. For more information, see GnuPG Download.

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.

   ```
   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:

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

   mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
   jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2fl74lmavr4Vg
   7K/KH8VHlq2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
   BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
   N+6pHBJrKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO
   2K7oVX8Oc3j5+ZilhL/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNIcO
   lFTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn
   5lBPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvRLsQRbcBG9qxX3
   rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq
   lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQq0osYXaQbvlnSWOoc1y/5Zb
   zizHTJIhLtUyls9WisP2s0emeHZicVMfW61EgPrJAiupgc7kyZvFt4YwfwARAQAB
   tCRBbWF6b24gRUNTIDxlY3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhwEEAECAAYF
   AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7
   Bs3pcPp8BV7BdRbs3sPLt5bV1+rkqOlw+0gZ4Q/ue/YbWtOAt4qY0OcEo0HgcnaX
   lsB827QIfZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmazA
   McWB4HUMNrhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF
   4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
   SReRXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
   bBCLzkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMZpVkUr+C252IaH1HZFEz+TvBVQM
   Y8OWWxmIJW+J6evjo3N1eO19UHv71jvoF8zljbI4bsL2c+QTJmOv7nRqzDQgCWyp
   Id/v2dUVVTk1j9omuLBBwNJzQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItenatK
   lEJQhYtyVXcBlh6Yn/wzNg2NWOwb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
   N2HqkTSQh77Z8KPKmyGopsmN/reMuilPdINb249nA0dzoN+nj+tTFOYCIaLaFyjs
   Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX
   gAAKCRC86dmkLVF4T9iFEACEnkm1dNXsWUx34R3c0vamHrPxvfkyI1FlEUen8D1h
   uX9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy
   uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZUgvXBL046JhG
   eM17+crgUyQeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFtgAsc9rk+
   YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1
   0X90J7cC1KOyhlEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlz3+6xBIm/qe
   bFyLUnf4WoiuOplAaJhK9pRY+XEnGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr
   KVHUq1TZD7cvMnnNKEELTUcKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
   XQ4zuF2IGCpvBFhYAlt5Un5zwqkwwQR3/n2kwAoDzonJcehDw/C/cGos5D0aIU7I
   K2X2aTD3+pA7Mx3IMe2hqmYqRt9X42yF1PIEVRneBRJ3HDezAgJrNh0GQWRQkhIx
   gz6/cTR+ekr5TptVszS9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
   1bkCDQRatUmrARAAxNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
   8uftYZqdC82KeSKhpHq7c8gMTMucIINtH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
   T+JIhZwdD8Mx2K+LVVVu/aWkNrfMuNwyDUciSI4D5QHa8T+F8fgN4OTpwYjirzel
   5yoICMr9hVcbzDNv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
   psiqXRYtVvYInEhLVrJ0aV6zHFoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
   WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
   t/MOdjXr1fjF4pThEi5u6WsuRdFwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
   2iHiOKIpQqLbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QuKE+I
   ```

```
R6jA0frUNt2jhiGG/F8RceXzohaaC/Cx7LUCUFWc0n7z32C9/Dtj7I1PMOacdZzz
bjJzRKO/ZDv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXPO0EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIACQUCWrVJqwIbDAAKCRC86dmkLVF4T+ZdD/9x/8APzgNJF3o3STrF
jvnV1ycyhWYGAeBJiu7wjsNWwzMFOv15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
X7DR0JszaH9wrYTxZLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt
Rwe/uwdibI0CagEzyX+2D3kTOlHO5XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZnl3LSmZyE0EQehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmcliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBWDhBPVPrruLIaenTtZEOsPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkfH6+6rbi1ujxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUUW
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcW+jsY9IhbEzqN5yQYGi4pVmDkY5vu
lXbJnbqPKpRXgM9BecV9AMbPgbDq/5LnHJJXg+G8YQOgp4lR/hC1TEFdIp5wM8AK
CWsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAr77kj7j2QR2SZeOSlFBvV7oS
mFeSNnz9xZssqrsm6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0mOmptr+xLLsKcaP7ogIXsyZnrEAEsvW8PnfayoiPCdc3cMCR/lTnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4blD32WqvyxESo7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSGm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTssoG+3OOcGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikAOlvE3/wgMqCXscbycbLjLD/bXIuFWo3rzoezeXjgi/DJx
jKBAyBTYO5nMcth1O9oaFd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvf
KeM1rO8Jm3iRAc5a/D0AEQEAAYkEPgQYAQIACQUCWrVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bXO1dQR4tKhhQ
P0LRqy6zlBY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOclTb9bBxnBc/4PFvHAbZW3QzusaZniNgkuxt6BTfloS
Of4inq71kjmGK+TlzQ6mUMQUg228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bhIIbQyEUB9gKCmUFaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphUl7T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLrWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA0OQxG
AHMOdJQQvBsmqMvuAdjkDWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vSjKpCNg3EIJSgqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8MelN9qr5KcKQLmfdfBNEyyceBhyVl0MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRyKToTjwOb73CGkBZZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivhoWwzjpfQKhwcU9lABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBXlR
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMii+mWra23EwjChaxpvjjcUH
5ilLc5Zq781aCYRygYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAX7kPMD3Lof4KldD
Q8ppQriUvxVo+4nPV6rpTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCEl7nbFrm0vIiA75Sa8KnywTDsyZsu3XcOcf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWHOis6bq8l9RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYlNr3lydh+dFHIEkH53HzQe6l88HEic
+0jVnLkCDQRa55wJARAAyLya2Lx6gyoWoJN1a6740q3o8e9d4KggQOfGMTCflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNJyjMNT1gAgrhPNB4HtXBXum2wS57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOIzQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSvOUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJIgjNvJscuUWb
+QEqJIRleJRhrO+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjziRceuJCJYox1cwsoq6jTE50GiNzcIxTn9xUc0UMKFeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm9lhkGwqTZgOlRX5eqPX+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6sYKqroYwBGgZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9zlW8qsA
EQEAAYkEPgQYAQgACQUCWuecCQIbAgIpCRC86dmkLVF4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykD/4tOLRHFHXuKUcxgGaubUcVtsFrwBKma1cYjqaPms8u
6Sk0wfGRI32G/GhOrp0Ts/MOkbObq6VLTh8N5Yc/53MEl8zQFw9Y5AmRoW4PZXER
ujs5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEFHYT8UTxwnGmXOvNkMM7md
xD3CV5q6VAte8WKBo/220II3fcQlc9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW/NN2gju2G3Lu/TlFUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUhQWeSNHI+fpe5oX6FALPT9JLDce8OZz1i
```

```
cZZ0MELP37mOOQun0AlmHm/hVzf0f311PtbzcqWaE51tJvgUR/nZFo6Ta3O5Ezhs
3VlEJNQ1Ijf/6DH87SxvAoRIARCuZd0qxBcDK0avpFzUtbJd24lRA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyz0WTDVmt
g0U1CO1ezokMSqkJ7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtfsCDvALCtqL0jduSkUo+RXcBItmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNtm9mw9ydIl1yjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6MlI2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxzlbNpS+zxt9r0MiLgcLyspWxSdmoYGZ6nQP
RO5Nm/ZVS+u2imPCRzNUZEMa+dlE6kHx0rS0dPiuJ4O7NtPeYDKkoQtNagspsDvh
cK7CSqAiKMq06UBTxqlTSRkm62eOCtcs3p3OeHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+45lcCfmcVt94TFNL5HwEUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNJ3bCLuc/jq7ysGq69xiKmTlUeXFm+aojcRO5i
zyShIRJZ0GZfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNxlf3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRGlU/LpNSefnvDFTtEIRcpOHc
bhayG0bk51Bd4mioOXnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj2O+X+J
qX2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==
=Gi5D
-----END PGP PUBLIC KEY BLOCK-----
```

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

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

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:

   ```
   curl -o ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-amd64-latest.asc
   ```

   - For Linux systems:

   ```
   curl -o ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-linux-amd64-latest.asc
   ```

   - For Windows systems:

   ```
   PS C:\> Invoke-WebRequest -OutFile ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-windows-amd64-latest.exe.asc
   ```

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:

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

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

Step 3: Apply Execute Permissions to the Binary

Apply execute permissions to the binary.

- For macOS and Linux systems:

```
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:

```
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. 512).

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.

```
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.

```
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. 514). 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`
   c. `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`
3. ECS config-attempts to fetch credentials from the default ECS profile.
4. Default AWS profile—Attempts to use credentials (`aws_access_key_id`, `aws_secret_access_key`) or `assume_role` (`role_arn`, `source_profile`) from the AWS profile name.
   a. `AWS_DEFAULT_PROFILE` environment variable (defaults to `default`).
5. EC2 instance role

The order of precedence for Region is:

1. Amazon ECS CLI flags:
   a. Region flag (`--region`)
   b. Cluster config flag (`--cluster-config`)
2. ECS config-attempts to fetch the Region from the default ECS profile.
3. Environment variables—Attempts to fetch the region from the following environment variables:
   a. `AWS_REGION`
   b. `AWS_DEFAULT_REGION`
4. AWS profile-attempts to use the region from the AWS profile name:
   a. `AWS_PROFILE` environment variable
   b. `AWS_DEFAULT_PROFILE` environment variable (defaults to `default`)

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 service with tasks 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. 506).
- Install and configure the AWS CLI. For more information, see AWS Command Line Interface.

Step 1: Create the Task Execution IAM Role

The Amazon ECS container agent makes calls to AWS APIs on your behalf, so it requires an IAM policy and role for the service to know that the agent belongs to you. This IAM role is referred to as a task execution IAM role. If you already have a task execution role created to use, you can skip this step. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

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:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Create the task execution role:

```
aws iam --region us-west-2 create-role --role-name ecsTaskExecutionRole --assume-role-
policy-document file://task-execution-assume-role.json
```

3. Attach the task execution role policy:

```
aws iam --region us-west-2 attach-role-policy --role-name ecsTaskExecutionRole --
policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

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. 512).

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:

```
ecs-cli configure --cluster tutorial --default-launch-type FARGATE --config-
name tutorial --region us-west-2
```

2. Create a CLI 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 tutorial-profile
```

Step 3: Create a Cluster and Configure the 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.

   ```
   ecs-cli up --cluster-config tutorial --ecs-profile tutorial-profile
   ```

   This command may take a few minutes to complete as your resources are created. The output of this command contains the VPC and subnet IDs that are created. Take note of these IDs as they are used later.
2. Using the AWS CLI, retrieve the default security group ID for the VPC. Use the VPC ID from the previous output:

   ```
   aws ec2 describe-security-groups --filters Name=vpc-id,Values=VPC_ID --region us-west-2
   ```

   The output of this command contains your security group ID, which is used in the next step.
3. Using AWS CLI, add a security group rule to allow inbound access on port 80:

   ```
   aws ec2 authorize-security-group-ingress --group-id security_group_id --protocol tcp --
   port 80 --cidr 0.0.0.0/0 --region us-west-2
   ```

Step 4: Create a Compose File

For this step, create a simple Docker compose file that creates a simple PHP web 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 `web` 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.

```
version: '3'
services:
  web:
    image: amazon/amazon-ecs-sample
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-west-2
        awslogs-stream-prefix: web
```

Note
If your account already contains a CloudWatch Logs log group named `tutorial` in the `us-west-2` Region, choose a unique name so the ECS CLI creates a new log group for this tutorial.

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:

```
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:
        - "security group ID"
      assign_public_ip: ENABLED
```

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 --ecs-profile tutorial-profile
```

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 --ecs-
profile tutorial-profile
```

Output:

```
Name                                              State    Ports
 TaskDefinition  Health
tutorial/0c2862e6e39e4eff92ca3e4f843c5b9a/web  RUNNING  34.222.202.55:80->80/tcp
 tutorial:1      UNKNOWN
```

In the above example, you can see the `web` 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 PHP web application. 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 0c2862e6e39e4eff92ca3e4f843c5b9a --follow --cluster-config tutorial
 --ecs-profile tutorial-profile
```

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 --ecs-
profile tutorial-profile
```

Now you should see two more containers in your cluster:

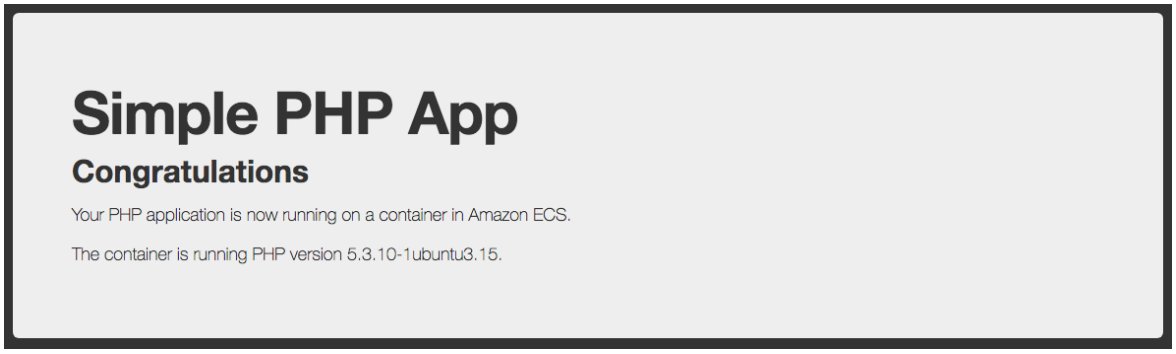
```
ecs-cli compose --project-name tutorial service ps --cluster-config tutorial --ecs-
profile tutorial-profile
```

Output:

```
Name                                                State    Ports
 TaskDefinition  Health
tutorial/0c2862e6e39e4eff92ca3e4f843c5b9a/web  RUNNING  34.222.202.55:80->80/tcp
 tutorial:1      UNKNOWN
tutorial/d9fbbc931d2e47ae928fcf433041648f/web  RUNNING  34.220.230.191:80->80/tcp
 tutorial:1      UNKNOWN
```

Step 9: View your Web Application

Enter the IP address for the task in your web browser and you should see a webpage that displays the **Simple PHP App** web application.



Step 10: 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 --ecs-
profile tutorial-profile
```

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 --ecs-profile tutorial-profile
```

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:

- Complete the steps in Setting Up with Amazon ECS (p. 7) and verify that your AWS user has either the permissions specified in the `AdministratorAccess` or the Amazon ECS First Run Wizard Permissions (p. 454) IAM policy example.
- Install the Amazon ECS CLI. For more information, see Installing the Amazon ECS CLI (p. 506).
- 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. 506).

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. 512).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration:

```
ecs-cli configure --cluster ec2-tutorial --default-launch-type EC2 --config-name ec2-
tutorial --region us-west-2
```

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-profile
```

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 --ecs-profile ec2-tutorial-profile
```

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 simple PHP web application. At this time, the Amazon ECS CLI supports Docker compose file syntax versions 1, 2, and 3. This tutorial uses Docker Compose version 3.

Here is the compose file, which you can call `docker-compose.yml`. The `web` container exposes port 80 to the container instance for inbound traffic to the web server. A logging configuration for the containers is also defined.

```
version: '3'
services:
  web:
    image: amazon/amazon-ecs-sample
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: ec2-tutorial
        awslogs-region: us-west-2
        awslogs-stream-prefix: web
```

When using Docker Compose version 3 format, the CPU and memory specifications must be specified separately. Create a file named `ecs-params.yml` with the following content:

```
version: 1
task_definition:
  services:
    web:
      cpu_shares: 100
      mem_limit: 524288000
```

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` *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 --ecs-profile ec2-
tutorial-profile
```

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 --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Output:

```
Name                                                State    Ports
 TaskDefinition  Health
ec2-tutorial/53c943778bf048ce954a6cb96425adeb/web  RUNNING  54.201.208.32:80->80/tcp
 ecscompose:1    UNKNOWN
```

In the above example, you can see the `web` container 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 PHP web application.

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 scale 2 --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Now you should see two more containers in your cluster:

```
ecs-cli ps --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Output:

```
Name                                                State    Ports
 TaskDefinition  Health
ec2-tutorial/53c943778bf048ce954a6cb96425adeb/web  RUNNING  54.201.208.32:80->80/tcp
 ecscompose:1     UNKNOWN
ec2-tutorial/9451480d53534a129fe6794941ad63dc/web  RUNNING  52.43.118.109:80->80/tcp
 ecscompose:1    UNKNOWN
```

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.

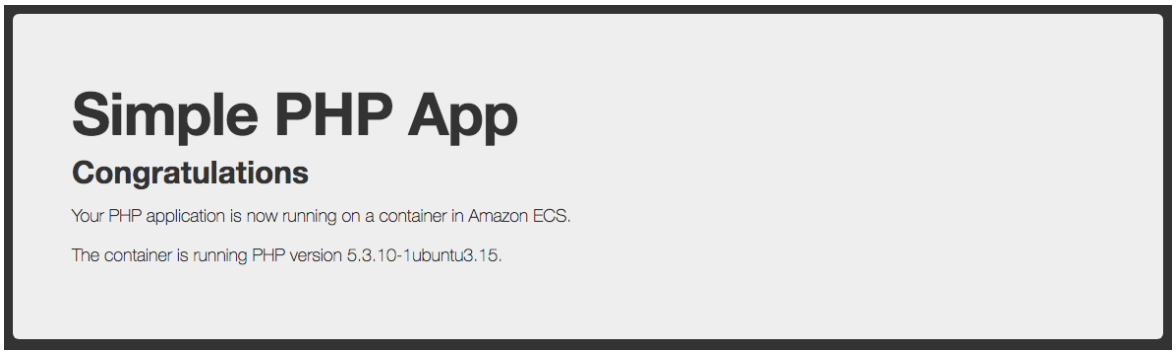
```
ecs-cli compose down --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Now you can create your service.

```
ecs-cli compose service up --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Step 8: View your Web Application

Enter the IP address for the task in your web browser and you should see a webpage that displays the **Simple PHP App** web application.



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 service rm --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Now, take down your cluster, which cleans up the resources that you created earlier with **ecs-cli up**.

```
ecs-cli down --force --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

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. 506).

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. 506).

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. 512).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration:

```
ecs-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[0034] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
INFO[0065] Created an ECS service                        service=backend
 taskDefinition="backend:1"
INFO[0066] Updated ECS service successfully              desiredCount=1
 serviceName=backend
INFO[0081] (service backend) has started 1 tasks: (task 824b5a76-8f9c-4beb-
a64b-6904e320630e).  timestamp="2018-09-12 00:00:26 +0000 UTC"
INFO[0157] Service status                                desiredCount=1 runningCount=1
 serviceName=backend
INFO[0157] ECS Service has reached a stable state        desiredCount=1 runningCount=1
 serviceName=backend
```

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
INFO[0033] Waiting for the Service Discovery Service to be created...
INFO[0034] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
INFO[0065] Created an ECS service                        service=frontend
 taskDefinition="frontend:1"
INFO[0066] Updated ECS service successfully              desiredCount=1
 serviceName=frontend
INFO[0081] (service frontend) has started 1 tasks: (task 824b5a76-8f9c-4beb-
a64b-6904e320630e).  timestamp="2018-09-12 00:00:26 +0000 UTC"
INFO[0157] Service status                                desiredCount=1 runningCount=1
 serviceName=frontend
INFO[0157] ECS Service has reached a stable state        desiredCount=1 runningCount=1
 serviceName=frontend
```

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[0002] 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 Service 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.

```
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:

```
ecs-cli --help
```

Note
Ensure that you are using the latest version of the Amazon ECS CLI. The latest version is 1.17.0. For release notes, see Changelog.

Available Commands

- ecs-cli (p. 526)
- ecs-cli configure (p. 527)
- ecs-cli up (p. 533)
- ecs-cli down (p. 541)
- ecs-cli scale (p. 543)
- ecs-cli ps (p. 544)
- ecs-cli push (p. 546)
- ecs-cli pull (p. 548)
- ecs-cli images (p. 550)
- ecs-cli license (p. 553)
- ecs-cli compose (p. 554)
- ecs-cli compose service (p. 566)
- ecs-cli logs (p. 588)
- ecs-cli check-attributes (p. 590)
- ecs-cli registry-creds (p. 592)
- ecs-cli local (p. 598)
- Using Docker Compose File Syntax (p. 605)
- Using Amazon ECS Parameters (p. 606)

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. 514).

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. 506).

Syntax

```
ecs-cli [--version] [subcommand] [--help]
```

Options

Name	Description
`--version, -v`	Prints the version information for the Amazon ECS CLI. Required: No
`--help, -h`	Show the help text for the specified command. Required: No

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. 527).

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. 533).

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. 533).

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. 541).

scale

Modifies the number of container instances in an ECS cluster. For more information, see ecs-cli scale (p. 543).

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. 588).

ps

Lists all of the running containers in an ECS cluster. For more information, see ecs-cli ps (p. 544).

push

Pushes an image to an Amazon ECR repository. For more information, see ecs-cli push (p. 546).

pull

Pulls an image from an ECR repository. For more information, see ecs-cli pull (p. 548).

images

Lists all of the running containers in an ECS cluster. For more information, see ecs-cli images (p. 550).

license

Prints the `LICENSE` files for the Amazon ECS CLI and its dependencies. For more information, see ecs-cli license (p. 553).

compose

Executes **docker-compose**–style commands on an ECS cluster. For more information, see ecs-cli compose (p. 554).

help

Shows the help text for the specified command.

ecs-cli configure

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. 506).

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. 529).
- 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. 529).

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

Name	Description
`--cluster, -c` *cluster_name*	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the **configure** command. Type: String Required: Yes
`--region, -r` *region*	Specifies the AWS Region to use. Defaults to the Region configured using either the **ecs-cli configure** or **aws configure** commands. Type: String Required: Yes
`--config-name` *config_name*	Specifies the name of this cluster configuration. This is the name that can be referenced in commands using the `--cluster-config` flag. If this option is omitted, then the name is set to `default`. Type: String Required: No
`--cfn-stack-name` *stack_name*	Specifies the stack name to add to the AWS CloudFormation stack that is created on **ecs-cli up**. **Important** It is not recommended to use this parameter. It is included to ensure backwards compatibility with previous versions of the ECS CLI. Type: String Default: `amazon-ecs-cli-setup-<cluster_name>` Required: No
`--default-launch-type` *launch_type*	Specifies the default launch type to use. Valid values are `FARGATE` or `EC2`. If not specified, no default launch type is used. For more information about launch types, see Amazon ECS Launch Types (p. 119). Type: String Required: No

Name	Description
`--help, -h`	Shows the help text for the specified command. Required: No

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: ecs-cli-demo
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. 506).

Syntax

ecs-cli configure default --config-name *config_name* [--help]

Options

Name	Description
`--config-name` *config_name*	Specifies the name of the cluster configuration to use by default in subsequent commands.

Name	Description
	Type: String Required: Yes
`--help, -h`	Shows the help text for the specified command. Required: No

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. 532).

> **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. 506).

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. 532).
- 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

Name	Description
`--profile-name` *profile_name*	Specifies the name of this ECS profile. This is the name that can be referenced in commands using the `--ecs-profile` flag. If this option is omitted, then the name is set to `default`. Type: String Required: Yes
`--access-key` *aws_access_key_id*	Specifies the AWS access key to use. If the `AWS_ACCESS_KEY_ID` environment variable is set when **ecs-cli configure profile** is run, then the AWS access key ID is set to the value of that environment variable. Type: String Required: Yes
`--secret-key` *aws_secret_access_key*	Specifies the AWS secret key to use. If the `AWS_SECRET_ACCESS_KEY` environment variable is set when **ecs-cli configure profile** is run, then the AWS secret access key is set to the value of that environment variable. Type: String Required: Yes
`--session-token` *token*	Specifies the AWS session token to use. If the `AWS_SESSION_TOKEN` environment variable if it is set when **ecs-cli configure profile** 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. Type: String Required: No
`--help, -h`	Shows the help text for the specified command. Required: No

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:

```
INFO[0000] Saved ECS CLI profile configuration default.
```

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. 506).

Syntax

ecs-cli configure profile default --profile-name *profile_name* [--help]

Options

Name	Description
`--profile-name` *profile_name*	Specifies the name of the ECS profile to be marked as default. Type: String Required: Yes
`--help, -h`	Shows the help text for the specified command. Required: No

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. 506).

Syntax

ecs-cli configure migrate [--force] [--help]

Options

Name	Description
`--force`	Omits the interactive description and confirmation step that normally occurs during the configuration file migration. Required: No
`--help, -h`	Shows the help text for the specified command. Required: No

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. 506).

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-
```

```
data string] [--instance-type instance_type] [--image-id ami_id] [--launch-
type launch_type] [--no-associate-public-ip-address] [--force] [--tags
key1=value1,key2=value2] [--cluster cluster_name] [--region region] [--empty]
[--verbose] [--help]
```

Options

Name	Description
`--capability-iam`	Acknowledges that this command may create IAM resources. **Note** This parameter is only supported with tasks that use the EC2 launch type. This parameter is required if you do not specify and instance profile name with `--instance-role`. You cannot specify both options. Required: No
`--keypair` *keypair_name*	Specifies the name of an existing Amazon EC2 key pair to enable SSH access to the EC2 instances in your cluster. **Note** This parameter is only supported with tasks that use the EC2 launch type. For more information about creating a key pair, see Setting Up with Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*. Type: String Required: No
`--size` *n*	Specifies the number of instances to launch and register to the cluster. **Note** This parameter is only supported with tasks that use the EC2 launch type. Type: Integer Default: 1 Required: No
`--azs` *availability_zone_1,availability_zone_2*	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 `--vpc` option. **Warning** Leaving this option blank can result in a failure to launch container instances when the randomly chosen zone is unavailable. Type: String

Name	Description
	Required: No
`--security-group` *security_group_id*[,*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

Name	Description
`--extra-user-data` *string*	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. 538). Type: String Required: No
`--instance-type` *instance_type*	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. **Note** This parameter is supported only with tasks that use the EC2 launch type. For more information on EC2 instance types, see Amazon EC2 Instances. Type: String Default: `t2.micro` Required: No
`--image-id` *ami_id*	Specifies the Amazon EC2 AMI ID to use for your container instances. 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 Systems Manager Parameter Store API during the cluster resource creation process. This requires the user account that you're using to have the required Systems Manager permissions. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 211). 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. **Note** This parameter is supported only with tasks that use the EC2 launch type. Type: String Default: The latest stable Amazon ECS–optimized AMI for the specified Region. Required: No

Name	Description
`--no-associate-public-ip-address`	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
`--force, -f`	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
`--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`. For more information, see Tagging Resources (p. 539). Type: Key value pairs Required: No
`--instance-role, -f` *`instance-profile-name`*	Specifies a custom IAM role name for instances in your cluster. A new instance profile will be created and attached to this role. **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
`--launch-type` *`launch_type`*	Specifies the launch type to use. Available options are `FARGATE` or `EC2`. For more information about launch types, see Amazon ECS Launch Types (p. 119). This overrides the default launch type stored in your cluster configuration. Type: String Required: No
`--verbose, --debug`	Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No

Name	Description
`--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
`--empty, -e`	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. Required: No
`--help, -h`	Shows the help text for the specified command. Required: No

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:

```
#!/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-shellscript \
  --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 Tagging Your Amazon ECS Resources (p. 407).

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. 184).
- 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
```

Output:

```
INFO[0001] Using recommended Amazon Linux AMI with ECS Agent 1.17.3 and Docker version
 17.12.1-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 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[0003] Waiting for your cluster resources to be created...
INFO[0003] Cloudformation stack status                   stackStatus="CREATE_IN_PROGRESS"
INFO[0066] Waiting for your cluster resources to be created...
INFO[0066] Cloudformation stack status                   stackStatus="CREATE_IN_PROGRESS"
VPC created: vpc-abcd1234
Subnets created: subnet-abcd1234
```

```
Subnets created: subnet-dcba4321
Cluster creation succeeded.
```

Creating an Empty Cluster

This example brings up an empty cluster named `ecs-cli-empty-demo` with no resources.

```
ecs-cli up --empty --cluster ecs-cli-empty-demo
```

Output:

```
INFO[0000] Created cluster                               cluster=ecs-cli-empty-demo
 region=us-east-1
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 Deleting a Cluster (p. 71) to 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. 506).

Syntax

```
ecs-cli down [--force] [--cluster cluster_name] [--region region] [--help]
```

Options

Name	Description
`--force, -f`	Acknowledges that this command permanently deletes resources and bypasses the confirmation prompt. Required: No
`--region, -r region`	Specifies the AWS Region to use. Defaults to the cluster configured using the **configure** command. Type: String

Name	Description
	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

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
INFO[0154] Deleted cluster
```

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
```

ecs-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. 506).

Syntax

```
ecs-cli scale --capability-iam --size n [--cluster cluster_name] [--region
region] [--help]
```

Options

Name	Description
`--capability-iam`	Acknowledges that this command may create IAM resources. Required: Yes
`--size` *n*	Specifies the number of instances to maintain in your cluster. Type: Integer Required: Yes
`--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

Name	Description
`--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

Examples

Example

This example scales the current cluster to two container instances.

```
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

Name	Description
`--desired-status` *status*	The container desired status to filter the container list results with. Required: No Valid values: `RUNNING` | `STOPPED`
`--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.

Name	Description
	Required: No

Examples

Example

This example shows the containers that are running in the cluster.

```
ecs-cli ps
```

Output:

```
Name                                                State    Ports
 TaskDefinition       Health
afd7f8a0-3813-4e1a-9d9e-ca7e9d1fcfbb/wordpress   RUNNING  36.253.177.221:80->80/tcp
 compose3:7           HEALTHY
dca67e02-68ca-4507-b194-a47239b5e7a9/wordpress   RUNNING  37.234.146.14:80->80/tcp
 healthcheck:3        UNKNOWN
dca67e02-68ca-4507-b194-a47239b5e7a9/redis       RUNNING
 healthcheck:3        HEALTHY
feb6e10e-3385-4c9b-a6cb-787cc8e90dda/sample-app  RUNNING  54.229.211.206:80->80/tcp
 tutorial-task-def:1  UNKNOWN
```

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, see Installing the Amazon ECS CLI (p. 506).

Syntax

ecs-cli push [--registry-id *registry_id*] [--tags *key1=value1,key2=value2*] [--region *region*] [--verbose] [--use-fips] *ECR_REPOSITORY*[:*TAG*] [--help]

Options

Name	Description
`--registry-id` *registry_id*	Specifies the Amazon ECR registry ID to which to push the image. By default, images are pushed to the current AWS account. Type: String Required: No
`--tags` *value*	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

Name	Description
	length of 256 characters. Tags use the following format: `key1=value1,key2=value2,key3=value3`. Type: Key value pairs Required: No
`--verbose, --debug`	Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No
`--use-fips`	Routes calls to Amazon ECR through FIPS endpoints. 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 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.

```
ecs-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. 506).

Syntax

```
ecs-cli pull [--registry-id registry_id] [--region region] [--verbose] [--use-fips] ECR_REPOSITORY[:TAG|@DIGEST] [--help]
```

Options

Name	Description
`--registry-id registry_id`	Specifies the Amazon ECR registry ID from which to pull the image. By default, images are pulled from the current AWS account. Required: No
`--verbose, --debug`	Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No
`--use-fips`	Routes calls to Amazon ECR through FIPS endpoints.

Name	Description
	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 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.

```
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. 506).

Syntax

```
ecs-cli images [--registry-id registry_id] [--tagged|--untagged] [--region
region] [--verbose] [--use-fips] [ECR_REPOSITORY] [--help]
```

Options

Name	Description
`--registry-id` *registry_id*	Specifies the Amazon ECR registry with which to list images. By default, images are listed for the current AWS account. Required: No
`--tagged`	Filters the result to show only tagged images. Required: No
`--untagged`	Filters the result to show only untagged images. Required: No
`--verbose, --debug`	Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No
`--use-fips`	Routes calls to Amazon ECR through FIPS endpoints. Required: No
`--region, -r` *region*	Specifies the AWS Region to use. Defaults to the cluster configured using the **configure** command. Type: String Required: No

Name	Description
`--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 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.

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
 sha256:404758ad8af94347fc8582fc8e30b6284f2b0751de29b2e755da212f80232fac   3 months ago
    203 MB
foobuntu                    latest
 sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8   4 days ago
    51.7 MB
ubuntu                      xenial
 sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8   4 days ago
    51.7 MB
ubuntu                      latest
 sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8   4 days ago
    51.7 MB
ubuntu                      <none>
 sha256:512e30a26d9fa3648dbccb9e78e9bab636e6022e2d80bd73c99177b21a0d3982   19 minutes ago
    268 MB
ubuntu                      trusty
 sha256:bd6d24e8fa3f5822146b2c94247976b87e6564195c3c180b67833e6ea699f7c2   18 minutes ago
    67.2 MB
ubuntu                      precise
 sha256:b38267a51fb4460699bc2bcdbb53d42fec697bb4e4f9a819df3e762cec393b2a   17 minutes ago
    40.1 MB
amazon-ecs-sample           latest
 sha256:bf04071a8edecc309f4d109ae36f24a5c272a115b6f7e636f77940059024d71c   2 weeks ago
    105 MB
golang                      latest
 sha256:137b22efee2df470b0cd28ebfc1ae583be0baf09334a5a882096193577d983ab   4 days ago
    266 MB
amazonlinux                 latest
 sha256:a59d563b5139deee8cb108bfb97bf3e9021b8ccea6dec8ff49733230cb2f0eca   4 days ago
    98.8 MB
awsbatch/fetch_and_run      latest
 sha256:543800007416d0ccff4f63643bb18eeff4b874ea772128efcdc231ff456a37fc   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
 sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8   4 days ago
    51.7 MB
ubuntu              latest
 sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8   4 days ago
    51.7 MB
ubuntu              <none>
 sha256:512e30a26d9fa3648dbccb9e78e9bab636e6022e2d80bd73c99177b21a0d3982   20 minutes ago
    268 MB
ubuntu              trusty
 sha256:bd6d24e8fa3f5822146b2c94247976b87e6564195c3c180b67833e6ea699f7c2   19 minutes ago
    67.2 MB
ubuntu              precise
 sha256:b38267a51fb4460699bc2bcdbb53d42fec697bb4e4f9a819df3e762cec393b2a   18 minutes ago
    40.1 MB
```

Example 3

This example lists all of the untagged images in an Amazon ECR registry.

```
ecs-cli images --untagged
```

Output:

```
REPOSITORY NAME     TAG                 IMAGE DIGEST
                       PUSHED AT           SIZE
ubuntu              <none>
 sha256:512e30a26d9fa3648dbccb9e78e9bab636e6022e2d80bd73c99177b21a0d3982   24 minutes ago
    268 MB
```

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. 506).

Syntax

```
ecs-cli license [--help]
```

Options

Name	Description
`--help, -h`	Shows the help text for the specified command. Required: No

Examples

Example

This example prints the license files.

```
ecs-cli license
```

Output:

```
Copyright 2015 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.
...
```

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. 605).

> **Note**
> To create Amazon ECS services with the Amazon ECS CLI, see ecs-cli compose service (p. 566).

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. 506).

Syntax

ecs-cli compose [--verbose] [--file *`compose_file`*] [--project-name *`project_name`*] [--task-role-arn *`task_role_arn`*] [--ecs-params *`ecs_params_file`*] [--registry-creds *`value`*] [--region *`region`*] [--cluster-config *`cluster_config_name`*] [--ecs-profile *`ecs_profile`*] [--aws-profile *`aws_profile`*] [--cluster *`cluster_name`*] [--help] [*`subcommand`*] [*`arguments`*] [--help]

Options

Name	Description
`--verbose, --debug`	Increases the verbosity of command output to aid in diagnostics. Required: No
`--file, -f` *`compose_file`*	Specifies the Docker Compose file to use. At this time, the latest version of the Amazon ECS CLI only supports the major versions of Docker Compose file syntax versions 1, 2, and 3. The version specified in the compose file must be the string `"1"`, `"1.0"`, `"2"`, `"2.0"`, `"3"`, or `"3.0"`. Docker Compose minor versions are not supported. If the `COMPOSE_FILE` environment variable is set when **ecs-cli compose** is run, the Docker Compose file is set to the value of that environment variable. Type: String

Name	Description
	Default: `./docker-compose.yml` Required: No
`--project-name, -p` *`project_name`*	Specifies the project name to use. If the `COMPOSE_PROJECT_NAME` environment variable is set when **ecs-cli compose** is run, the project name is set to the value of that environment variable. Type: String Default: The current directory name. Required: No
`--task-role-arn` *`role_value`*	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
`--ecs-params` *`ecs_params_file`*	Specifies the ECS parameters that aren't native to Docker Compose files. For more information, see Using Amazon ECS Parameters (p. 606). Default: `./ecs-params.yml` Required: No
`--registry-creds` *`value`*	Specifies the Amazon ECS registry credentials file to use. Defaults to the latest output file from the `ecs-cli registry-creds up` command, if one exists. For more information, see ecs-cli registry-creds (p. 592). Default: `./ecs-registry-creds_`*`[TIMESTAMP]`*`.yml` 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

Name	Description
`--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

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. 557).

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. 560).

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. 563).

service [*subcommand*]

Creates an ECS service from your Compose file. For more information, see ecs-cli compose service (p. 566).

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.

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. 506).

Syntax

ecs-cli compose create [--region *region*] [--cluster-config *cluster_config_name*] [--ecs-profile *ecs_profile*] [--aws-profile *aws_profile*] [--cluster *cluster_name*] [--launch-type *launch_type*] [--create-log-groups] [--tags *key1=value1,key2=value2*] [--help]

Options

Name	Description
`--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

Name	Description
`--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
`--launch-type` *launch_type*	Specifies the launch type to use. Available options are `FARGATE` or `EC2`. For more information about launch types, see Amazon ECS Launch Types (p. 119). This overrides the default launch type stored in your cluster configuration. Type: String Required: No
`--create-log-groups`	Creates the CloudWatch log groups specified in your Compose files. 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. 581). 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 Tagging Your Amazon ECS Resources (p. 407).

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`:

```
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`:

```
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. 506).

Syntax

ecs-cli compose start [--region *region*] [--cluster-config *cluster_config_name*] [--ecs-profile *ecs_profile*] [--aws-profile *aws_profile*] [--cluster *cluster_name*] [--launch-type *launch_type*] [--create-log-groups] [--help]

Options

Name	Description
`--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
`--launch-type` *launch_type*	Specifies the launch type to use. Available options are `FARGATE` or `EC2`. For more information about launch types, see Amazon ECS Launch Types (p. 119). This overrides the default launch type stored in your cluster configuration. Type: String Required: No
`--create-log-groups`	Creates the CloudWatch log groups specified in your Compose files. Required: No
`--help, -h`	Shows the help text for the specified command.

Name	Description
	Required: No

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`:

```
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`:

```
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 --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
```

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. 506).

Syntax

ecs-cli compose up [--region *region*] [--cluster-config *cluster_config_name*] [--ecs-profile *ecs_profile*] [--aws-profile *aws_profile*] [--cluster *cluster_name*] [--launch-type *launch_type*] [--create-log-groups] [--force-update] [--tags *key1=value1,key2=value2*] [--disable-ecs-managed-tags] [--help]

Options

Name	Description
`--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

Name	Description
`--launch-type` *launch_type*	Specifies the launch type to use. Available options are `FARGATE` or `EC2`. For more information about launch types, see Amazon ECS Launch Types (p. 119). This overrides the default launch type stored in your cluster configuration. Type: String Required: No
`--create-log-groups`	Creates the CloudWatch log groups specified in your Compose files. Required: No
`--force-update`	Forces the relaunching of the tasks. 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. 581). Type: Key value pairs Required: No
`--disable-ecs-managed-tags`	Disable the Amazon ECS managed tags. For more information, see Tagging Your Resources for Billing (p. 409). 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 Tagging Your Amazon ECS Resources (p. 407).

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. 409).

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`:

```
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`:

```
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. 605).

> **Note**
> To run tasks with the Amazon ECS CLI instead of creating services, see ecs-cli compose (p. 554).

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. 506).

Syntax

```
ecs-cli compose [--verbose] [--file compose_file] [--project-name project_name]
[--task-role-arn task_role_arn] [--ecs-params ecs_params_file] [--registry-
creds value] [--region region] [--cluster-config cluster_config_name] [--ecs-
profile ecs_profile] [--aws-profile aws_profile] [--cluster cluster_name] [--
help] service [subcommand] [arguments] [--help]
```

Options

Name	Description
`--verbose, --debug`	Increases the verbosity of command output to aid in diagnostics. Required: No
`--file, -f` *`compose_file`*	Specifies the Docker Compose file to use. At this time, the latest version of the Amazon ECS CLI only supports the major versions of Docker Compose file syntax versions 1, 2, and 3. The version specified in the compose file must be the string `"1"`, `"1.0"`, `"2"`, `"2.0"`, `"3"`, or `"3.0"`. Docker Compose minor versions are not supported. If the `COMPOSE_FILE` environment variable is set when **ecs-cli compose** is run, the Docker Compose file is set to the value of that environment variable. Type: String Default: `./docker-compose.yml` Required: No

Name	Description
`--project-name, -p` *project_name*	Specifies the project name to use. If the `COMPOSE_PROJECT_NAME` environment variable is set when **ecs-cli compose** is run, the project name is set to the value of that environment variable. Type: String Default: The current directory name. Required: No
`--task-role-arn` *role_value*	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
`--ecs-params` *ecs_params_file*	Specifies the ECS parameters that aren't native to Docker Compose files. For more information, see Using Amazon ECS Parameters (p. 606). Default: `./ecs-params.yml` Required: No
`--registry-creds` *value*	Specifies the Amazon ECS registry credentials file to use. Defaults to the latest output file from the `ecs-cli registry-creds up` command, if one exists. For more information, see ecs-cli registry-creds (p. 592). Default: `./ecs-registry-creds_`*[TIMESTAMP]*`.yml` 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

Name	Description
`--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

Available Subcommands

The **ecs-cli compose service** 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 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. 569).

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. 574).

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. 576).

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. 582).

scale

Scales the desired count of the service to the specified count. For more information, see ecs-cli compose service scale (p. 583).

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. 585).

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. 586).

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

ecs-cli compose service create [--deployment-max-percent *n*] [--deployment-min-healthy-percent *n*] [--load-balancer-name *value*] [--target-group-arn *value*] [--target-groups "targetGroupArn=arn:elasticloadbalancing:*region*:*aws_account_id*:targetgroup/*target_group_name_1*,containerName=*container_name*,containerPort=*container_port*"**] [--container-name *value*] [--container-port *value*] [--role *value*] [--launch-type *launch_type*] [--health-check-grace-period *integer*] [--create-log-groups] [--enable-service-discovery] [--vpc *value*] [--private-dns-namespace *value*] [--private-dns-namespace-id *value*] [--public-dns-namespace *value*] [--public-dns-namespace-id *value*] [--sd-container-name *value*] [--sd-container-port *value*] [--dns-ttl *value*] [--dns-type *value*] [--healthcheck-custom-config-failure-threshold *value*] [--scheduling-strategy *value*] [--tags *key1=value1,key2=value2*] [--disable-ecs-managed-tags] [--help]**

Options

Name	Description
`--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. 344). 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. 344). 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. **Deprecated**, use `target-groups`. Required: No
`--target-groups`	Specifies one or more target groups to be registered with the service. Specify the Amazon Resource Name (ARN) of the target group, the container name and the container port for each group to register. A single `--targetgroups` flag specifies only one target group. Add additional `--target-groups` flags to add additional target groups. Type: Strings

Name	Description
	For more information see Registering Multiple Target Groups with a Service (p. 393). 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. **Deprecated**, use `target-groups`. 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. **Deprecated**, use `target-groups`. 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. **Deprecated**, use `target-groups`. 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

Name	Description
`--private-dns-namespace`	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
`--private-dns-namespace-id`	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
`--public-dns-namespace`	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
`--public-dns-namespace-id`	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
`--sd-container-name`	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.
`--sd-container-port`	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.
`--dns-ttl`	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
`--dns-type`	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

Name	Description
`--healthcheck-custom-config-failure-threshold`	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
`--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. 339). • `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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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. 339). **Note** Fargate tasks do not support the `DAEMON` scheduling strategy. For more information, see Service scheduler concepts (p. 338). 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. 581). 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. 409). 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. 376). 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. 382).

- 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. 522).

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 Tagging Your Amazon ECS Resources (p. 407).

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 propogated 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. 409).

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

Name	Description
`--timeout` *value*	Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count

Name	Description
	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
`--create-log-groups`	Creates the CloudWatch log groups specified in your Compose files. Required: No
`--force-deployment`	Forces a new deployment of the service. 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

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

ecs-cli compose service up [--deployment-max-percent *n*] [--deployment-min-healthy-percent *n*] [--load-balancer-name *value*] [--target-group-arn *value*] [--target-groups "targetGroupArn=arn:elasticloadbalancing:*region*:*aws_account_id*:targetgroup/*target_group_name_1*,containerName=*container_name*,containerPort=*container_port*"] **[--container-name *value*] [--container-port *value*] [--role *value*] [--health-check-grace-period *integer*] [--timeout *value*] [--launch-type *launch_type*] [--create-log-groups] [--force-deployment] [--enable-service-discovery] [--vpc *value*] [--private-dns-namespace *value*] [--private-dns-namespace-id *value*] [--public-dns-namespace *value*] [--public-dns-namespace-id *value*] [--sd-container-name *value*] [--sd-container-port *value*] [--dns-ttl *value*] [--dns-type *value*] [--healthcheck-custom-config-failure-threshold *value*] [--update-service-discovery] [--scheduling-strategy *value*] [--tags *key1=value1,key2=value2*] [--disable-ecs-managed-tags] [--help]**

Options

Name	Description
`--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. 344). 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. 344). 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. **Deprecated**, use `target-groups`. Required: No
`--target-groups`	Specifies one or more target groups to be registered with the service. Specify the Amazon Resource Name (ARN) of the target group, the container name and the container port for each group to register. A single `--targetgroups` flag specifies only one target group. Add additional `--target-groups` flags to add additional target groups. Type: Strings

Name	Description
	For more information see Registering Multiple Target Groups with a Service (p. 393). 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. **Deprecated**, use `target-groups`. 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. **Deprecated**, use `target-groups`. 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. **Deprecated**, use `target-groups`. 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
`--force-deployment`	Forces a new deployment of the service. Required: No
`--enable-service-discovery`	Specifies whether to enable service discovery for this service. Required: No

Name	Description
`--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
`--private-dns-namespace`	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
`--private-dns-namespace-id`	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
`--public-dns-namespace`	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
`--public-dns-namespace-id`	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
`--sd-container-name`	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.
`--sd-container-port`	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.
`--dns-ttl`	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

Name	Description
`--dns-type`	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
`--healthcheck-custom-config-failure-threshold`	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
`--update-service-discovery`	If specified, this enables the service discovery service settings for `--dns-ttl` and `--healthcheck-custom-config-failure-threshold` to be updated. Required: No
`--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. 339). • `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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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. 339). **Note** Fargate tasks do not support the `DAEMON` scheduling strategy. For more information, see Service scheduler concepts (p. 338). Type: String Valid values: `REPLICA` | `DAEMON` Default value: `REPLICA` Required: No

Name	Description
`--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. 581). Type: Key value pairs Required: No
`--disable-ecs-managed-tags`	Disable the Amazon ECS managed tags. For more information, see Tagging Your Resources for Billing (p. 409). 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. 376). 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. 382).

- 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. 522).

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 Tagging Your Amazon ECS Resources (p. 407).

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 propogated 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. 409).

Examples

Example 1

This example brings up an Amazon ECS service with the project name `hello-world` from the `hello-world.yml` compose file.

```
ecs-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"
INFO[0000] Updated ECS service successfully              desiredCount=1
 serviceName=ecscompose-service-hello-world
INFO[0015] (service ecscompose-service-hello-world) has started 1 tasks: (task
 682dc22f-8bfa-4c28-b6f8-3a916bd8f86a).  timestamp=2017-08-18 21:16:00 +0000 UTC
INFO[0060] Service status                                desiredCount=1 runningCount=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.

```
ecs-cli compose -f nginx-compose.yml service up --target-group-arn
 arn:aws:elasticloadbalancing:us-east-1:aws_account_id:targetgroup/ecs-cli-
alb/9856106fcc5d4be8 --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.

```
ecs-cli compose -f nginx-compose.yml service up --target-group-arn
 arn:aws:elasticloadbalancing:us-east-1:aws_account_id:targetgroup/ecs-cli-
alb/9856106fcc5d4be8 --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

Name	Description
`--desired-status` *status*	The container desired status to filter the container list results with. Required: No Valid values: `RUNNING` | `STOPPED`
`--region, -r` *region*	Specifies the AWS Region to use. Defaults to the cluster configured using the **configure** command. Type: String

Name	Description
	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

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

Name	Description
`--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. 344). Default value: `200`

Name	Description
	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. 344). Default value: `100` Required: No
`--timeout` *value*	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` 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

Name	Description
`--help, -h`	Shows the help text for the specified command. Required: No

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-a8a9-80328c302b89).  timestamp=2017-08-18 21:17:44 +0000 UTC
INFO[0075] Service status                                desiredCount=2 runningCount=2
 serviceName=ecscompose-service-hello-world
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

Name	Description
`--timeout` *value*	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` Required: No

Name	Description
`--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

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.

ecs-cli compose service rm|delete|down [--timeout *value*] [--delete-namespace] [--help]

Options

Name	Description
`--timeout` *value*	Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count

Name	Description
	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
`--delete-namespace`	If specified, the private namespace created with either the **compose service create** or **compose service up** commands is deleted. 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

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.

```
ecs-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
```

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. 506).

Syntax

```
ecs-cli logs --task-id task_id [--task-def task_definition] [--follow]
[--filter-pattern search_string] [--since n_minutes] [--start-time
2006-01-02T15:04:05+07:00] [--end-time 2006-01-02T15:04:05+07:00] [--
timestamps] [--help]
```

Options

Name	Description
`--task-id` *task_id*	Prints the logs for this ECS task. Type: String Required: Yes
`--task-def` *task_definition*	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.

Name	Description
	Type: String Required: No
`--follow`	Specifies if the logs should be streamed. Required: No
`--filter-pattern` *search_string*	Specifies the substring to search for within the logs. Type: String Required: No
`--since` *n*	Returns logs newer than a relative duration in minutes. Can't be used with `--start-time`. Type: Integer Required: No
`--start-time` *timestamp*	Returns logs after a specific date (format: RFC 3339. Example: `2006-01-02T15:04:05+07:00`). Can't be used with `--since` flag. Required: No
`--end-time` *timestamp*	Returns logs before a specific date (format: RFC 3339. Example: `2006-01-02T15:04:05+07:00`). Cannot be used with `--follow`. Required: No
`--timestamps`	Specifies if timestamps are shown on each line in the log output. 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

Name	Description
`--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

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. 506).

Syntax

```
ecs-cli check-attributes [--task-def task_definition] [--container-instances
value] [--help]
```

Options

Name	Description
`--task-def` *task_definition*	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. Type: String Required: No
`--container-instances` *value*	A list of container instance IDs or full ARN entries to check if all required attributes are available for the Task Definition to RunTask. 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

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:

```
Container Instance                   Missing Attributes
28c5abd2-360e-41a0-81d8-0afca2d08d9b  com.amazonaws.ecs.capability.logging-driver.fluentd
45510138-f24f-47c6-a418-71c46dd51f88  None
```

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 (p. 160).

> **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. 506).

Syntax

```
ecs-cli registry-creds [--region region] [--cluster-config cluster_config_name]
[--ecs-profile ecs_profile] [--aws-profile aws_profile] [--help] [subcommand]
[arguments] [--help]
```

Options

Name	Description
`--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

Name	Description
	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

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. 593).

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. 506).

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

Name	Description
./creds_input_file.yml	Specifies the values related to private registry authentication. For more information, see Using Private Registry Authentication (p. 594).

Name	Description
	Required: Yes
`--role-name` *value*	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. 482). **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
`--update-existing-secrets`	Specifies whether existing secrets should be updated with new credential values. Required: No
`--no-role`	If specified, no task execution role is created. Required: No
`--no-output-file`	If specified, no output file for use with `compose` is created. Required: No
`--output-dir` *value*	The directory where the output file should be created. If none specified, the file is created in the current working directory. 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`. For more information, see Tagging Resources (p. 595). Type: Key value pairs Required: No
`--help, -h`	Shows the help text for the specified command. Required: No

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 follow 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. 554) or ecs-cli compose service (p. 566).

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 Tagging Your Amazon ECS Resources (p. 407).

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 `database` in the Docker compose file.

```
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] New credential secret created:
 arn:aws:secretsmanager:region:aws_account_id:secret:amazon-ecs-cli-setup-dockerhub-VeDqXm
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] Created new task execution role policy arn:aws:iam::aws_account_id:policy/
amazon-ecs-cli-setup-bugBashRole-policy-20181023T210805Z
INFO[0000] Attached AWS managed policy arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy to role ecsTaskExecutionRole
INFO[0001] Attached new policy arn:aws:iam::aws_account_id:policy/amazon-ecs-cli-setup-
bugBashRole-policy-20181023T210805Z to role ecsTaskExecutionRole
INFO[0001] Writing registry credential output to new file C:\Users\brandejo\regcreds
\regCredTest\ecs-registry-creds_20181023T210805Z.yml
```

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. 554) or ecs-cli compose service (p. 566).

The following is an example output file:

```
version: "1"
registry_credential_outputs:
  task_execution_role: secretsTaskExecutionRole
  container_credentials:
    dockerhub:
      credentials_parameter: arn:aws:secretsmanager:region:aws_account_id:secret:amazon-
ecs-cli-setup-dockerhub-bbHiEk
      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.

```
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.

```
version: '1'
registry_credentials:
  dockerhub:
    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.

ecs-cli local

Runs your Amazon ECS tasks locally by creating a Docker Compose file from 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. 506).

Syntax

```
ecs-cli local [subcommand] [arguments] [--help]
```

Options

Name	Description
`--region, -r` *region*	Specifies the AWS Region to use. Defaults to the Region configured using the **configure** command. 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
`--help, -h`	Shows the help text for the specified command. Required: No

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 a Docker Compose file from an Amazon ECS task definition. For more information, see ecs-cli local create (p. 599).

up

Runs containers locally from an Amazon ECS task definition. For more information, see ecs-cli local up (p. 600).

down

Stops and removes locally running containers. For more information, see ecs-cli local down (p. 602).

ps

Lists locally running containers. For more information, see ecs-cli local ps (p. 603).

help

Shows the help text for the specified command.

ecs-cli local create

Creates a Docker Compose file from 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. 506).

Syntax

ecs-cli local create [--task-def-file *filename*] [--task-def-remote *value*] [--force] [--output *output_file*]

Options

Name	Description
`--task-def-file` *filename*	Specifies the filename that contains the task definition JSON to convert to a Docker Compose file. If one is not specified, the ECS CLI will look for a file named `task-definition.json` in the current directory. Type: JSON Required: No
`--task-def-remote` *value*	Specifies the full Amazon Resource Name (ARN) or family:revision of the task definition to convert to a Docker Compose file. If you specify a task definition family without a revision, the latest revision is used. Type: string Required: No
`--force`	Overwrites any existing Docker Compose output file without prompting for confirmation.

Name	Description
`--output` *output_file*	Specifies the local filename to write the Docker Compose file to. If one is not specified, the default is `docker-compose.local.yml`. Type: string Required: No
`--help, -h`	Shows the help text for the specified command. Required: No

Examples

Create a Docker Compose file from a local JSON file

This example creates a Docker Compose file from a local JSON file containing an Amazon ECS task definition.

```
ecs-cli local create --task-def-file task-definition.json
```

Output:

```
INFO[0000] Successfully wrote docker-compose.ecs-local.yml
INFO[0000] Successfully wrote docker-compose.ecs-local.override.yml
```

Create a Docker Compose file from a remote task definition

This example creates a Docker Compose file from the latest revision of an Amazon ECS task definition named `hello-world`.

```
ecs-cli local create --task-def-remote hello-world
```

Output:

```
INFO[0000] Successfully wrote docker-compose.ecs-local.yml
INFO[0000] Successfully wrote docker-compose.ecs-local.override.yml
```

ecs-cli local up

Runs containers locally from an Amazon ECS task definition. By default this commands looks for a task definition JSON file named `task-definition.json` in the current directory. If the task definition file does not exist, then one must be specified using one of the `--task-def` options described below. This command also creates a local Docker Compose file as specified in the `--output` option prior to running the containers.

> **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. 506).

Syntax

ecs-cli local create up [--task-def-compose *filename*] [--task-def-file *filename*] [--task-def-remote *value*] [--force] [--output *output_file*] [--override *filename*]

Options

Name	Description
`--task-def-compose` *filename*	Specifies the Docker Compose file to run locally. Type: string Required: No
`--task-def-file` *filename*	Specifies the task definition JSON file to run locally. If one is not specified, the ECS CLI will look for a file named `task-definition.json` in the current directory. Type: string Required: No
`--task-def-remote` *value*	Specifies the full Amazon Resource Name (ARN) or family:revision of the task definition to convert to a Docker Compose file. If you specify a task definition family without a revision, the latest revision is used. Type: string Required: No
`--force`	Overwrites any existing Docker Compose output file without prompting for confirmation.
`--output` *output_file*	Specifies the local filename to write the Docker Compose file to. If one is not specified, the default is `docker-compose.local.yml`. If the output file already exists, the CLI will prompt you with an overwrite request. Type: string Required: No
`--override` *filename*	Specifies the local Docker Compose override filename to use. Type: string Required: No

Examples

Run containers locally from a local task definition JSON file

This example runs the containers locally that are defined in a local task definition file named `hello-world.json`.

```
ecs-cli local create --task-def-file hello-world.json
```

Output:

```
INFO[0001] Successfully wrote docker-compose.ecs-local.yml
```

```
INFO[0002] Successfully wrote docker-compose.ecs-local.override.yml
INFO[0002] The network ecs-local-network already exists
INFO[0002] The amazon-ecs-local-container-endpoints container already exists with ID
 5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1
INFO[0002] Started container with ID
 5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1
INFO[0002] Using docker-compose.ecs-local.yml, docker-compose.ecs-local.override.yml files
 to start containers
Compose out: Found orphan containers (downloads_httpd_1) for this project. If you removed
 or renamed this service in your compose file, you can run this command with the --remove-
orphans flag to clean it up.
Creating downloads_simple-app_1 ... done
```

Run containers locally from a remote task definition

This example runs the containers locally that are defined in the latest revision of an Amazon ECS task definition named `hello-world`.

```
ecs-cli local up --task-def-remote hello-world
```

Output:

```
INFO[0000] Reading task definition from hello-world

INFO[0002] Successfully wrote docker-compose.ecs-local.yml
INFO[0004] Successfully wrote docker-compose.ecs-local.override.yml
INFO[0004] The network ecs-local-network already exists
INFO[0005] The amazon-ecs-local-container-endpoints container already exists with ID
 5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1
INFO[0005] Started container with ID
 5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1
INFO[0005] Using docker-compose.ecs-local.yml, docker-compose.ecs-local.override.yml files
 to start containers
Compose out: Found orphan containers (downloads_httpd_1) for this project. If you removed
 or renamed this service in your compose file, you can run this command with the --remove-
orphans flag to clean it up.
Creating downloads_hello-world_1 ... done
```

ecs-cli local down

Stops and removes locally running containers.

> **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. 506).

Syntax

ecs-cli local down [--task-def-file *filename*] [--task-def-remote *task_definition_ARN_family*] [--all]

Options

Name	Description
`--task-def-file` *filename*	Stop and remove all running containers matching the task definition filename. If both `--task-def-file` and `--`

Name	Description
	`task-def-remote` are omitted, the ECS CLI defaults to `task-definition.json`. Type: string Required: No
`--task-def-remote` *value*	Stops and remove all running containers matching the specified task definition Amazon Resource Name (ARN) or family:revision. If you specify a task definition family without a revision, the latest revision is used. Type: string Required: No
`--all`	Stops and removes all locally running containers. Type: string Required: No

Examples

Stop a locally running container

This example stops a locally running container that is using the `hello-world.json` task definition file.

```
ecs-cli local down --task-def-file hello-world.json
```

Output:

```
INFO[0000] Stop and remove 1 container(s)
INFO[0011] Stopped container with id 9df4c584d905
INFO[0011] Removed container with id 9df4c584d905
INFO[0011] The network ecs-local-network has no more running tasks
INFO[0012] Stopped container with name amazon-ecs-local-container-endpoints
INFO[0012] Removed container with name amazon-ecs-local-container-endpoints
INFO[0012] Removed network with name ecs-local-network
```

Stop all locally running containers

This example stops all locally running containers.

```
ecs-cli local down --all
```

ecs-cli local ps

Lists locally running containers.

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. 506).

Syntax

ecs-cli local ps [--task-def-file *filename*] [--task-def-remote *value*] [--all] [--json]

Options

Name	Description
`--task-def-file` *filename*	Lists all locally running containers matching the task definition filename. If both `--task-def-file` and `--task-def-remote` are omitted, the ECS CLI defaults to `task-definition.json`. Type: string Required: No
`--task-def-remote` *value*	Lists all running containers matching the task definition Amazon Resource Name (ARN) or family:revision. If you specify a task definition family without a revision, the latest revision is used. Type: string Required: No
`--all`	Lists all locally running containers. Type: string Required: No
`--json`	Sets the output to JSON format. Type: string Required: No

Examples

List all locally running containers

This example lists all locally running containers.

```
ecs-cli local ps --all
```

Output:

```
CONTAINER ID        IMAGE               STATUS              PORTS                  NAMES
                TASKDEFINITION
9df4c584d905        httpd:2.4           Up 15 seconds       0.0.0.0:80->80/tcp     /
downloads_simple-app_1   /Users/brandejo/Downloads/task-definition.json
```

List all locally running containers using a specified task definition file

This example lists the locally running containers using the `hello-world.json` task definition.

```
ecs-cli local ps --task-def-file hello-world.json
```

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. 554) and ecs-cli compose service (p. 566).

At this time, the latest version of the Amazon ECS CLI only supports the major versions of Docker Compose file syntax versions 1, 2, and 3. The version specified in the compose file must be the string `"1"`, `"1.0"`, `"2"`, `"2.0"`, `"3"`, or `"3.0"`. Docker Compose minor versions are not supported.

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 3.0 format, `cpu_shares` should be specified in the `ecs-params.yml` file. For more information, see Using Amazon ECS Parameters (p. 606).
- `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 1.0 and 2 only)
- `extra_hosts`
- `healthcheck` (Compose file version 3.0 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. 606).
- `hostname`
- `image`
- `labels`
- `links` (not valid for tasks using the Fargate launch type)

- `log_driver` (Compose file version 1.0 only)
- `log_opt` (Compose file version 1.0 only)
- `logging` (Compose file version 2.0 and 3.0)
  - `driver`
  - `options`
- `mem_limit` (in bytes)

  Note
  If you're using the Compose version 3.0 format, `mem_limit` should be specified in the `ecs-params.yml` file. For more information, see Using Amazon ECS Parameters (p. 606).
- `mem_reservation` (in bytes)

  Note
  If you're using the Compose version 3.0 format, `mem_reservation` should be specified in the `ecs-params.yml` file. For more information, see Using Amazon ECS Parameters (p. 606).
- `ports`
- `privileged` (not valid for tasks using the Fargate launch type)
- `read_only`
- `security_opt`
- `shm_size` (Compose file version 1.0 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 1.0 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:
  ecs_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
      firelens_configuration:
        type: string
        options:
          enable-ecs-log-metadata: boolean
      secrets:
        - value_from: string
          name: string
  docker_volumes:
      - name: string
        scope: string
        autoprovision:
        driver: string
        driver_opts: boolean
           string: string
        labels:
           string: string
  efs_volumes:
      - name: string
        filesystem_id: string
        root_directory: string
        transit_encryption: string
        transit_encryption_port: int64
        access_point: string
        iam: string
  placement_constraints:
      - type: string
        expression: 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:
```

```
      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. 85).
- `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. 84).
- `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. 482).
- `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.

CPU value	Memory value (MiB)
256 (.25 vCPU)	512 (0.5GB), 1024 (1GB), 2048 (2GB)
512 (.5 vCPU)	1024 (1GB), 2048 (2GB), 3072 (3GB), 4096 (4GB)
1024 (1 vCPU)	2048 (2GB), 3072 (3GB), 4096 (4GB), 5120 (5GB), 6144 (6GB), 7168 (7GB), 8192 (8GB)
2048 (2 vCPU)	Between 4096 (4GB) and 16384 (16GB) in increments of 1024 (1GB)
4096 (4 vCPU)	Between 8192 (8GB) and 30720 (30GB) in increments of 1024 (1GB)

  For more information, see Task size (p. 114).
- `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. 160).
  - `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. 121).
- `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. 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`.
- `firelens_configuration` – This parameter allows you to define a log configuration using the `awsfirelens` log driver. This is used to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Custom Log Routing (p. 150).
  - `type` – The log router type to use. Supported options are `fluentbit` and `fluentd`.
  - `options` – The log router options to use. This will depend on the destination you are routing your logs to. For more information, see Custom Log Routing (p. 150).
- `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. 163).
  - `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`, `autoprovision`, `driver`, `driver_opts` and `labels` correspond with the Docker volume configuration fields in a task definition. For more information, see DockerVolumeConfiguration 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.
- `efs_volumes` – This parameter enables you to mount Amazon EFS file systems. The `name` and `filesystem_id` keys are required.
  - `name` – 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.
  - `filesystem_id` – The ID of the file system for which to create the mount target.
  - `root_directory` – The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume will be used. Specifying / will have the same effect as omitting this parameter.
  - `transit_encryption` – Whether or not to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. This parameter is required if IAM is enabled or an access point ID is specified. Valid values are `ENABLED or DISABLED. DISABLED` is the default.
  - `transit_encryption_port` – The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you do not specify a transit encryption port, it will use the port selection strategy that the Amazon EFS mount helper uses. This parameter is required if `transit_encryption` is enabled.

  - `access_point` – The ID of the Amazon EFS access point to use. If an access point is specified, the root directory value will be relative to the directory set for the access point. If specified, `transit_encryption` must be enabled.
  - `iam` – Whether or not to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, `transit encryption` must be enabled in the EFSVolumeConfiguration. Valid values are `ENABLED or DISABLED. DISABLED` is the default.
- `placement_constraints` – This parameter allows you to specify a list of constraints on task placement within the task definition. For more information, see TaskDefinitionPlacementConstraint in the *Amazon Elastic Container Service API Reference*. It is optional if you are using the EC2 launch type. It is not supported if using the Fargate launch type.

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).

- `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 if using the Fargate launch type. For more information, see Amazon ECS Task Placement (p. 320).

  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. 522).

Amazon ECS service quotas

The following table provides the default service quotas, also referred to as limits, for Amazon ECS for an AWS account. For more information on the service quotas for other AWS services that you can use with Amazon ECS, such as Elastic Load Balancing and Auto Scaling, see AWS service quotas in the *Amazon Web Services General Reference*.

Adjustable service quotas

The following service quotas are adjustable.

To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

Service quota	Description	Default quota value
Clusters per account	The maximum number of clusters per Region, per account.	10,000
Container instances per cluster	The maximum number of container instances per cluster.	2,000
Services per cluster	The maximum number of services per cluster.	1,000
Tasks using the EC2 launch type per service (the desired count)	The maximum number of tasks using the EC2 launch type per service (the desired count). This limit applies to both standalone tasks and tasks launched as part of a service.	1,000
Tasks using the Fargate launch type or the `FARGATE` capacity provider, per Region, per account	The maximum number of tasks using the Fargate launch type or the `FARGATE` capacity provider, per Region. This limit applies to both standalone tasks and tasks launched as part of a service.	100
Fargate Spot tasks, per Region, per account	The maximum number of tasks using the `FARGATE_SPOT` capacity provider, per Region.	250
Public IP addresses for tasks using the Fargate launch type	The maximum number of public IP addresses used by tasks using the Fargate launch type, per Region.	100

Nonadjustable service quotas

The following are other limitations for Amazon ECS that cannot be changed.

Note
The quotas for Fargate task storage are nonadjustable and are dependent on the platform version used by the task. For more information, see Fargate Task Storage (p. 125).

Service quota	Description	Default quota value
Tasks launched (`count`) per **run-task**	The maximum number of tasks that can be launched per `RunTask` API action.	10
Container instances per **start-task**	The maximum number of container instances specified in a `StartTask` API action.	10
Revisions per task definition family	The maximum number of revisions per task definition family. Deregistering a task definition revision does not exclude it from being included in this limit.	1,000,000
Task definition size limit	The maximum size, in KiB, of a task definition.	32
Task definition max containers	The maximum number of containers definitions within a a task definition.	10
Subnets specified in an `awsvpcConfiguration`	The maximum number of subnets specified within an `awsvpcConfiguration`.	16
Security groups specified in an `awsvpcConfiguration`	The maximum number of security groups specified within an `awsvpcConfiguration`.	5
Classic Load Balancers per service	The maximum number of Classic Load Balancers per service.	1
Tags per resource	The maximum number of tags per resource. This applies to tasks, services, task definitions, clusters, and container instances.	50

Using Service Quotas

Amazon Elastic Container Service (Amazon ECS) has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. Service quotas are also referred to as limits. For more information, see What Is Service Quotas? in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of all of the Amazon ECS service quotas.

To view Amazon ECS service quotas (AWS Management Console)

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.

3. From the **AWS services** list, search for and select **Amazon Elastic Container Service (Amazon ECS)**.

   In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.
4. To view additional information about a service quota, such as the description, choose the quota name.

To request a quota increase, see Requesting a Quota Increase in the *Service Quotas User Guide*.

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. 615)
- Batch Jobs (p. 617)

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. 615)
- Service Discovery (p. 616)
- Authorization and Secrets Management (p. 616)
- Logging (p. 616)
- Continuous Integration and Continuous Deployment (p. 616)

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. 416). 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. 424).

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. 395).

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. 489) 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. 237).

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. 143).

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.

Savings Plans and Amazon ECS

Savings Plans are a pricing model that offer significant savings on AWS usage. This pricing model offers lower prices on Amazon EC2 instances usage, regardless of instance family, size, operating system, tenancy or AWS Region, and also apply to AWS Fargate usage. You commit to a consistent amount of usage, in USD per hour, for a term of 1 or 3 years, and receive a lower price for that usage. For more information, see the Savings Plans User Guide.

Each type of usage has an On-Demand rate and a Savings Plans price. For example, if you commit to $10/hour of compute usage, you'll receive Savings Plans prices on all usage up to $10. Any usage beyond the commitment is charged at On-Demand rates.

Savings Plans are available for 1 year or 3 year terms. You have the choice between **All Upfront**, **Partial upfront**, or **No upfront** payment options.

The following Savings Plans types are available:

- **Compute Savings Plans** provide the most flexibility and provide a discount of up to 66 percent. These plans automatically apply to your Amazon EC2 instance usage, regardless of instance family (for example, M5, C5), instance sizes (c5.large, c5.xlarge), Region (US-East-1, US-East-2), operating system (Windows, Linux), or tenancy (Dedicated, default, dedicated host). They also apply to your Fargate usage. You can move your Amazon ECS tasks from using Amazon EC2 to Fargate at any time and continue to receive the discounted rate provided by your Compute Savings Plan.
- **Amazon EC2 Instance Savings Plans** provide a discount of up to 72 percent, in exchange for a commitment to a specific instance type and Region. You can change your instance size within the instance type (for example, from `c5.xlarge` to `c5.2xlarge`) or the operating system (for example, from Windows to Linux) and continue to receive the discounted rate provided by your Amazon EC2 Instance Savings Plan.

To get started, see Get Started with Savings Plans in the *Savings Plans User Guide*.

Amazon Elastic Container Service on AWS Outposts

AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. Amazon ECS on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see the AWS Outposts User Guide.

Prerequisites

The following are the prerequisites for using Amazon ECS on AWS Outposts:

- You must have installed and configured an Outpost in your on-premises data center.
- You must have a reliable network connection between your Outpost and its AWS Region.
- You must have sufficient capacity of instance types available in your Outpost.
- All Amazon ECS container instances must have Amazon ECS container agent 1.33.0 or later.

Limitations

The following are the limitations of using Amazon ECS on Outposts:

- Amazon Elastic Container Registry, AWS Identity and Access Management, Application Load Balancer, Network Load Balancer, Classic Load Balancer, and Amazon Route 53 run in the AWS Region, not on Outposts. This will increase latencies between these services and the containers.
- AWS Fargate is not available on AWS Outposts.

Network Connectivity Considerations

The following are network connectivity considerations for AWS Outposts:

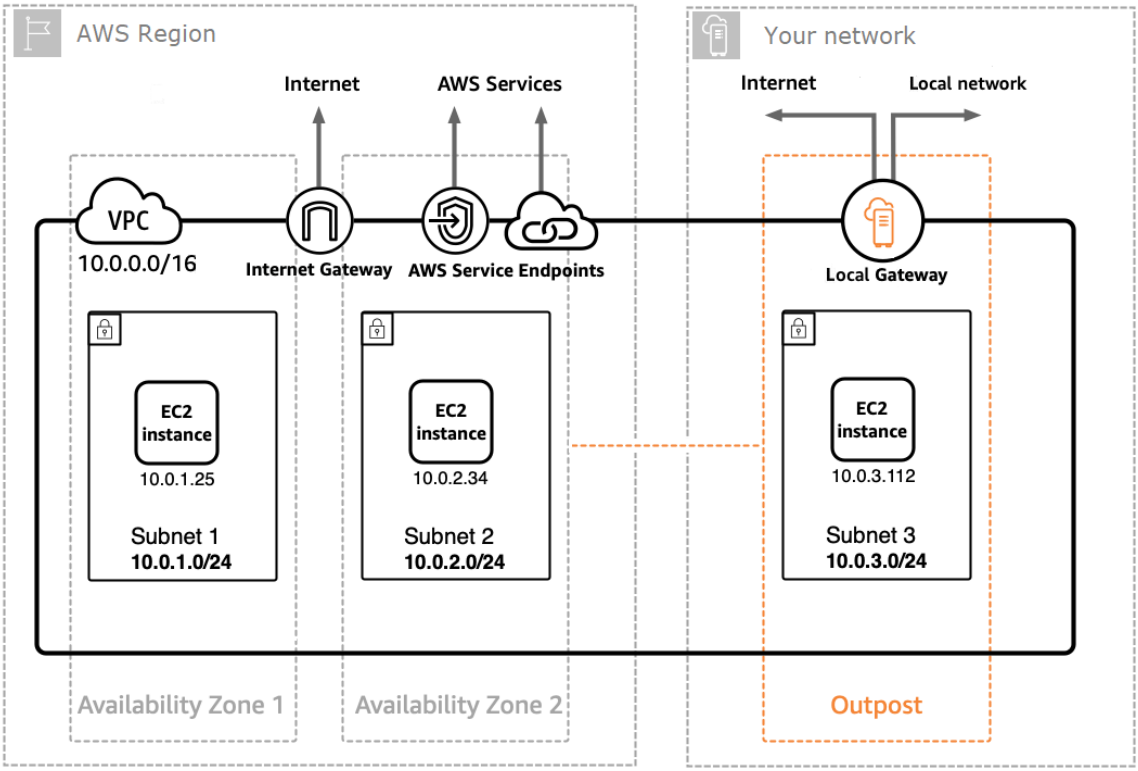
- If network connectivity between your Outpost and its AWS Region is lost, your clusters will continue to run. However, you cannot create new clusters or take new actions on existing clusters until connectivity is restored. In case of instance failures, the instance will not be automatically replaced. The CloudWatch Logs agent will be unable to update logs and event data.
- We recommend that you provide reliable, highly available, and low latency connectivity between your Outpost and its AWS Region.

Creating an Amazon ECS Cluster on an Outpost

Creating an Amazon ECS cluster on an Outpost is similar to creating an Amazon ECS cluster in the AWS Cloud. When you create an Amazon ECS cluster on an Outpost, you must specify a subnet associated with your Outpost.

An Outpost is an extension of an AWS Region, and you can extend an Amazon VPC in an account to span multiple Availability Zones and any associated Outposts. When you configure your Outpost,

you associate a subnet with it to extend your Regional VPC environment to your on-premises facility. Instances on an Outpost appear as part of your Regional VPC, similar to an Availability Zone with associated subnets.



AWS CLI

To create an Amazon ECS cluster on an Outpost with the AWS CLI, specify a security group and a subnet that is associated with your Outpost.

To create a subnet associated with your Outpost.

```
aws ec2 create-subnet \
    --cidr-block 10.0.3.0/24 \
    --vpc-id vpc-xxxxxxxx \
    --outpost-arn arn:aws:outposts:us-west-2:123456789012:outpost/op-xxxxxxxxxxxxxxxxx \
    --availability-zone-id usw2-az1
```

The following example creates an Amazon ECS cluster on an Outpost.

1. Create a role and policy with rights on Outpost.

```
aws iam create-role --role-name ecsRole \
    -assume-role-policy-document file://ecs-policy.json
aws iam put-role-policy --role-name ecsRole --policy-name ecsRolePolicy \
    --policy-document file://role-policy.json
```

2. Create an IAM instance profile with rights on Outpost.

```
aws iam create-instance-profile --instance-profile-name outpost
aws iam add-role-to-instance-profile --instance-profile-name outpost \
```

```
    --role-name ecsRole
```

3. Create a VPC.

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16
```

4. Create a security group for the container instances, specifying the proper CIDR range for the Outpost. (This step is different for AWS Outposts.)

```
aws ec2 create-security-group --group-name MyOutpostSG
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
    --port 22 --cidr 10.0.3.0/24
aws ec2 authorize-security-group-ingress ----group-name MyOutpostSG ----protocol tcp \
    --port 80 --cidr 10.0.3.0/24
```

5. Create the Cluster.
6. Define the Amazon ECS container agent environment variables to launch the instance into the cluster created in the previous step and define any necessary tags.

```
#! /bin/bash
cat << 'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_IMAGE_PULL_BEHAVIOR=prefer-cached
ECS_CONTAINER_INSTANCE_TAGS={"environment": "Outpost"}
EOF
```

Note
In order to avoid delays caused by pulling container images from Amazon ECR in the Region, use image caches. To do this, each time a task is run, configure the Amazon ECS agent to default to using the cached image on the instance itself by setting `ECS_IMAGE_PULL_BEHAVIOR` to `prefer-cached`.

7. Create the container instance, specifying the VPC and subnet for the Outpost where this instance should run and an instance type that is available on the Outpost. (This step is different for AWS Outposts.)

```
aws ec2 run-instances --count 1 --image-id ami-xxxxxxxx --instance-type c5.large \
    --key-name aws-outpost-key --subnet-id subnet-xxxxxxxxxxxxxxxxx \
    --iam-instance-profile Name outpost --security-group-id sg-xxxxxx \
    --associate-public-ip-address --user-data file://userdata.txt
```

Note
This command is also used when adding additional instances to the cluster. Any containers deployed in the cluster will be placed on that specific Outpost.

8. Register a task definition.

```
aws ecs register-task-definition ----cli-input-json file://ecs-task.json
```

9. Run the task or create the service.

   Run the task

```
aws ecs run-task --cluster mycluster --count 1 --task-definition outpost-app:1
```

   Create the service

```
aws ecs create-service --cluster mycluster --service-name outpost-service \
```

```
    --task-definition outpost-app:1 --desired-count 1
```

Getting started with AWS App Mesh and Amazon ECS

AWS App Mesh is a service mesh based on the Envoy proxy that helps you monitor and control services. App Mesh standardizes how your services communicate, giving you end-to-end visibility into and helping to ensure high-availability for your applications. App Mesh gives you consistent visibility and network traffic controls for every service in an application. For more information, see the AWS App Mesh User Guide.

This topic helps you use AWS App Mesh with an actual service that is running on Amazon ECS. This tutorial covers basic features of App Mesh. To learn more about other features that you'll, but that aren't used when completing this tutorial, see the topics for virtual nodes, virtual services, virtual routers, routes, and the Envoy proxy.

Scenario

To illustrate how to use App Mesh with Amazon ECS, assume that you have an application with the following characteristics:

- Includes two services named `serviceA` and `serviceB`.
- Both services are registered to a namespace named `apps.local`.
- `ServiceA` communicates with `serviceB` over HTTP/2, port 80.
- You've already deployed version 2 of `serviceB` and registered it with the name `serviceBv2` in the `apps.local` namespace.

You have the following requirements:

- You want to send 75 percent of the traffic from `serviceA` to `serviceB` and 25 percent of the traffic to `serviceBv2` to ensure that `serviceBv2` is bug free before you send 100 percent of the traffic from `serviceA` to it.
- You want to be able to easily adjust the traffic weighting so that 100 percent of the traffic goes to `serviceBv2` once it's proven to be reliable. Once all traffic is being sent to `serviceBv2`, you want to deprecate `serviceB`.
- You don't want to have to change any existing application code or service discovery registration for your actual services to meet the previous requirements.

To meet your requirements, you've decided to create an App Mesh service mesh with virtual services, virtual nodes, a virtual router, and a route. After implementing your mesh, you update the task definitions for your services to use the Envoy proxy. Once updated, your services communicate with each other through the Envoy proxy rather than directly with each other.

Prerequisites

App Mesh supports Linux services that are registered with DNS, AWS Cloud Map, or both. To use this getting started guide, we recommend that you have three existing services that are registered with DNS. You can create a service mesh and its resources even if the services don't exist, but you can't use the mesh until you have deployed actual services.

For more information about service discovery on Amazon ECS, see Service Discovery. To create an Amazon ECS service with service discovery, see Tutorial: Creating a Service Using Service Discovery.

The remaining steps assume that the actual services are named `serviceA`, `serviceB`, and `serviceBv2` and that all services are discoverable through a namespace named `apps.local`.

Step 1: Create a mesh and virtual service

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*. A virtual service is an abstraction of an actual service. For more information, see Virtual Services in the *AWS App Mesh User Guide*.

Create the following resources:

- A mesh named `apps`, since all of the services in the scenario are registered to the `apps.local` namespace.
- A virtual service named `serviceb.apps.local`, since the virtual service represents a service that is discoverable with that name, and you don't want to change your code to reference another name. A virtual service named `servicea.apps.local` is added in a later step.

You can use the AWS Management Console or the AWS CLI version 1.18.71 or higher or 2.0.17 or higher to complete the following steps. If using the AWS CLI, use the `aws --version` command to check your installed AWS CLI version. If you don't have version 1.18.71 or higher or 2.0.17 or higher installed, then you must install or update the AWS CLI. Select the tab for the tool that you want to use.

AWS Management Console

1. Open the App Mesh console first-run wizard at https://console.aws.amazon.com/appmesh/get-started.
2. For **Mesh name**, enter **apps**.
3. For **Virtual service name**, enter **serviceb.apps.local**.
4. To continue, choose **Next**.

AWS CLI

1. Create a mesh with the `create-mesh` command.

```
aws appmesh create-mesh --mesh-name apps
```

2. Create a virtual service with the `create-virtual-service` command.

```
aws appmesh create-virtual-service --mesh-name apps --virtual-service-name
 serviceb.apps.local --spec {}
```

Step 2: Create a virtual node

A virtual node acts as a logical pointer to an actual service. For more information, see Virtual Nodes *in the AWS App Mesh User Guide*.

Create a virtual node named `serviceB`, since one of the virtual nodes represents the actual service named `serviceB`. The actual service that the virtual node represents is discoverable through `DNS` with a hostname of `serviceb.apps.local`. Alternately, you can discover actual services using AWS Cloud

Map. The virtual node will listen for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported, as are health checks. You will create virtual nodes for `serviceA` and `serviceBv2` in a later step.

AWS Management Console

1. For **Virtual node name**, enter **`serviceB`**.
2. For **Service discovery method**, choose **DNS** and enter **`serviceb.apps.local`** for **DNS hostname**.
3. Under **Listener configuration**, choose **http2** for **Protocol** and enter **`80`** for **Port**.
4. To continue, choose **Next**.

AWS CLI

1. Create a file named `create-virtual-node-serviceb.json` with the following contents:

```
{
    "meshName": "apps",
    "spec": {
        "listeners": [
            {
                "portMapping": {
                    "port": 80,
                    "protocol": "http2"
                }
            }
        ],
        "serviceDiscovery": {
            "dns": {
                "hostname": "serviceB.apps.local"
            }
        }
    },
    "virtualNodeName": "serviceB"
}
```

2. Create the virtual node with the create-virtual-node command using the JSON file as input.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-
serviceb.json
```

Step 3: Create a virtual router and route

Virtual routers route traffic for one or more virtual services within your mesh. For more information, see Virtual Routers and Routes in the *AWS App Mesh User Guide*.

Create the following resources:

- A virtual router named `serviceB`, since the `serviceB.apps.local` virtual service doesn't initiate outbound communication with any other service. Remember that the virtual service that you created previously is an abstraction of your actual `serviceb.apps.local` service. The virtual service sends traffic to the virtual router. The virtual router will listen for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported.
- A route named `serviceB`. It will route 100 percent of its traffic to the `serviceB` virtual node. You'll change the weight in a later step once you've added the `serviceBv2` virtual node. Though not covered in this guide, you can add additional filter criteria for the route and add a retry policy to cause

the Envoy proxy to make multiple attempts to send traffic to a virtual node when it experiences a communication problem.

AWS Management Console

1. For **Virtual router name,** enter **`serviceB`**.
2. Under **Listener configuration**, choose **http2** for **Protocol** and specify **`80`** for **Port**.
3. For **Route name**, enter **`serviceB`**.
4. For **Route type**, choose **http2**.
5. For **Virtual node name** under **Route configuration**, select `serviceB` and enter **`100`** for **Weight**.
6. To continue, choose **Next**.

AWS CLI

1. Create a virtual router.
   a. Create a file named `create-virtual-router.json` with the following contents:

```
{
    "meshName": "apps",
    "spec": {
        "listeners": [
            {
                "portMapping": {
                    "port": 80,
                    "protocol": "http2"
                }
            }
        ]
    },
    "virtualRouterName": "serviceB"
}
```

   b. Create the virtual router with the create-virtual-router command using the JSON file as input.

```
aws appmesh create-virtual-router --cli-input-json file://create-virtual-
router.json
```

2. Create a route.
   a. Create a file named `create-route.json` with the following contents:

```
{
    "meshName" : "apps",
    "routeName" : "serviceB",
    "spec" : {
        "httpRoute" : {
            "action" : {
                "weightedTargets" : [
                    {
                        "virtualNode" : "serviceB",
                        "weight" : 100
                    }
                ]
            },
            "match" : {
                "prefix" : "/"
            }
```

```
        }
    },
    "virtualRouterName" : "serviceB"
}
```

b. Create the route with the create-route command using the JSON file as input.

```
aws appmesh create-route --cli-input-json file://create-route.json
```

Step 4: Review and create

Review the settings against the previous instructions.

AWS Management Console

Choose **Edit** if you need to make changes in any section. Once you're satisfied with the settings, choose **Create mesh**.

The **Status** screen shows you all of the mesh resources that were created. You can see the created resources in the console by selecting **View mesh**.

AWS CLI

Review the settings of the mesh you created with the describe-mesh command.

```
aws appmesh describe-mesh --mesh-name apps
```

Review the settings of the virtual service that you created with the describe-virtual-service command.

```
aws appmesh describe-virtual-service --mesh-name apps --virtual-service-name
 serviceb.apps.local
```

Review the settings of the virtual node that you created with the describe-virtual-node command.

```
aws appmesh describe-virtual-node --mesh-name apps --virtual-node-name serviceB
```

Review the settings of the virtual router that you created with the describe-virtual-router command.

```
aws appmesh describe-virtual-router --mesh-name apps --virtual-router-name serviceB
```

Review the settings of the route that you created with the describe-route command.

```
aws appmesh describe-route --mesh-name apps \
    --virtual-router-name serviceB  --route-name serviceB
```

Step 5: Create additional resources

To complete the scenario, you need to:

- Create one virtual node named `serviceBv2` and another named `serviceA`. Both virtual nodes listen for requests over HTTP/2 port 80. For the `serviceA` virtual node, configure a backend of

`serviceb.apps.local`, since all outbound traffic from the `serviceA` virtual node is sent to the virtual service named `serviceb.apps.local`. Though not covered in this guide, you can also specify a file path to write access logs to for a virtual node.

- Create one additional virtual service named `servicea.apps.local`, which will send all traffic directly to the `serviceA` virtual node.
- Update the `serviceB` route that you created in a previous step to send 75 percent of its traffic to the `serviceB` virtual node and 25 percent of its traffic to the `serviceBv2` virtual node. Over time, you can continue to modify the weights until `serviceBv2` receives 100 percent of the traffic. Once all traffic is sent to `serviceBv2`, you can deprecate the `serviceB` virtual node and actual service. As you change weights, your code doesn't require any modification, because the `serviceb.apps.local` virtual and actual service names don't change. Recall that the `serviceb.apps.local` virtual service sends traffic to the virtual router, which routes the traffic to the virtual nodes. The service discovery names for the virtual nodes can be changed at any time.

AWS Management Console

1. In the left navigation pane, select **Meshes**.
2. Select the `apps` mesh that you created in a previous step.
3. In the left navigation pane, select **Virtual nodes**.
4. Choose **Create virtual node**.
5. For **Virtual node name**, enter **`serviceBv2`**, for **Service discovery method**, choose **DNS**, and for **DNS hostname**, enter **`servicebv2.apps.local`**.
6. For **Listener configuration**, select **http2** for **Protocol** and enter **`80`** for **Port**.
7. Choose **Create virtual node**.
8. Choose **Create virtual node** again. Enter **`serviceA`** for the **Virtual node name**. For **Service discovery method**, choose **DNS**, and for **DNS hostname**, enter **`servicea.apps.local`**.
9. For **Enter a virtual service name** under **New backend**, enter **`servicea.apps.local`**.
10. Under **Listener configuration**, choose **http2** for **Protocol**, enter **`80`** for **Port**, and then choose **Create virtual node**.
11. In the left navigation pane, select **Virtual routers** and then select the `serviceB` virtual router from the list.
12. Under **Routes**, select the route named `ServiceB` that you created in a previous step, and choose **Edit**.
13. Under **Targets**, **Virtual node name**, change the value of **Weight** for `serviceB` to **`75`**.
14. Choose **Add target**, choose `serviceBv2` from the drop-down list, and set the value of **Weight** to **`25`**.
15. Choose **Save**.
16. In the left navigation pane, select **Virtual services** and then choose **Create virtual service**.
17. Enter **`servicea.apps.local`** for **Virtual service name**, select **Virtual node** for **Provider**, select `serviceA` for **Virtual node**, and then choose **Create virtual service.**

AWS CLI

1. Create the `serviceBv2` virtual node.

   a. Create a file named `create-virtual-node-servicebv2.json` with the following contents:

```
{
    "meshName": "apps",
    "spec": {
        "listeners": [
```

```
            {
                "portMapping": {
                    "port": 80,
                    "protocol": "http2"
                }
            }
        ],
        "serviceDiscovery": {
            "dns": {
                "hostname": "serviceBv2.apps.local"
            }
        }
    },
    "virtualNodeName": "serviceBv2"
}
```

b. Create the virtual node.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-servicebv2.json
```

2. Create the `serviceA` virtual node.

a. Create a file named `create-virtual-node-servicea.json` with the following contents:

```
{
   "meshName" : "apps",
   "spec" : {
      "backends" : [
         {
            "virtualService" : {
               "virtualServiceName" : "serviceb.apps.local"
            }
         }
      ],
      "listeners" : [
         {
            "portMapping" : {
               "port" : 80,
               "protocol" : "http2"
            }
         }
      ],
      "serviceDiscovery" : {
         "dns" : {
            "hostname" : "servicea.apps.local"
         }
      }
   },
   "virtualNodeName" : "serviceA"
}
```

b. Create the virtual node.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-servicea.json
```

3. Update the `serviceb.apps.local` virtual service that you created in a previous step to send its traffic to the `serviceB` virtual router. When the virtual service was originally created, it didn't send traffic anywhere, since the `serviceB` virtual router hadn't been created yet.

a. Create a file named `update-virtual-service.json` with the following contents:

```
{
   "meshName" : "apps",
   "spec" : {
      "provider" : {
         "virtualRouter" : {
            "virtualRouterName" : "serviceB"
         }
      }
   },
   "virtualServiceName" : "serviceb.apps.local"
}
```

b. Update the virtual service with the update-virtual-service command.

```
aws appmesh update-virtual-service --cli-input-json file://update-virtual-
service.json
```

4. Update the `serviceB` route that you created in a previous step.

a. Create a file named `update-route.json` with the following contents:

```
{
   "meshName" : "apps",
   "routeName" : "serviceB",
   "spec" : {
      "http2Route" : {
         "action" : {
            "weightedTargets" : [
               {
                  "virtualNode" : "serviceB",
                  "weight" : 75
               },
               {
                  "virtualNode" : "serviceBv2",
                  "weight" : 25
               }
            ]
         },
         "match" : {
            "prefix" : "/"
         }
      }
   },
   "virtualRouterName" : "serviceB"
}
```

b. Update the route with the update-route command.

```
aws appmesh update-route --cli-input-json file://update-route.json
```

5. Create the `serviceA` virtual service.

a. Create a file named `create-virtual-servicea.json` with the following contents:

```
{
   "meshName" : "apps",
   "spec" : {
      "provider" : {
         "virtualNode" : {
            "virtualNodeName" : "serviceA"
         }
```

```
      }
   },
   "virtualServiceName" : "servicea.apps.local"
}
```

b. Create the virtual service.

```
aws appmesh create-virtual-service --cli-input-json file://create-virtual-
servicea.json
```

Mesh summary

Before you created the service mesh, you had three actual services named `servicea.apps.local`, `serviceb.apps.local`, and `servicebv2.apps.local`. In addition to the actual services, you now have a service mesh that contains the following resources that represent the actual services:

- Two virtual services. The proxy sends all traffic from the `servicea.apps.local` virtual service to the `serviceb.apps.local` virtual service through a virtual router.
- Three virtual nodes named `serviceA`, `serviceB`, and `serviceBv2`. The Envoy proxy uses the service discovery information configured for the virtual nodes to look up the IP addresses of the actual services.
- One virtual router with one route that instructs the Envoy proxy to route 75 percent of inbound traffic to the `serviceB` virtual node and 25 percent of the traffic to the `serviceBv2` virtual node.

Step 6: Update services

After creating your mesh, you need to complete the following tasks:

- Authorize the Envoy proxy that you deploy with each Amazon ECS task to read the configuration of one or more virtual nodes. For more information about how to authorize the proxy, see Proxy authorization.
- Update each of your existing Amazon ECS task definitions to use the Envoy proxy.

Credentials

The Envoy container requires AWS Identity and Access Management credentials for signing requests that are sent to the App Mesh service. For Amazon ECS tasks deployed with the Amazon EC2 launch type, the credentials can come from the instance role or from a task IAM role. Amazon ECS tasks deployed with the Fargate launch type do not have access to the Amazon EC2 metadata server that supplies instance IAM profile credentials. To supply the credentials, you must attach an IAM task role to any tasks deployed with the Fargate launch type. If a task is deployed with the Amazon EC2 launch type and access is blocked to the Amazon EC2 metadata server, as described in the *Important* annotation in IAM Role for Tasks, then a task IAM role must also be attached to the task. The role that you assign to the instance or task must have an IAM policy attached to it as described in Proxy authorization.

Update task definitions

You can update your task definitions by using the AWS Management Console or by modifying the JSON file for a task definition. The following steps only show updating the `taskB` task for the scenario. You also need to update the `taskBv2` and `taskA` tasks by changing the values appropriately. Select the method you prefer to use to update the task definition.

AWS Management Console

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. From the pre-requisites and previous steps, you might have task definitions named `taskA`, `taskB`, and `taskBv2`. Select `taskB` and choose **Create new revision**.
5. On the **Create new revision of Task Definition** page, make the following changes to enable App Mesh integration.

   a. For **Service Integration**, to configure the parameters for App Mesh integration choose **Enable App Mesh integration** and then do the following:

      i. For **Application container name**, choose the container name to use for the App Mesh application. This container must already be defined within the task definition.

      ii. For **Envoy image**, enter one of the following images:

      - All supported Regions other than `me-south-1` and `ap-east-1`. You can replace *`us-west-2`* with any Region other than `me-south-1` and `ap-east-1`.

```
840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
```

      - `me-south-1` Region:

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
```

      - `ap-east-1` Region:

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
```

      iii. For **Mesh name**, choose the App Mesh service mesh to use. In this topic, the name of the mesh that was created is `apps`.

      iv. For **Virtual node name**, choose the App Mesh virtual node to use. For example, for the `taskB` task, you would choose the `serviceB` virtual node that you created in a previous step.

      v. The value for **Virtual node port** is pre-populated with the listener port that you specified when you created the virtual node.

      vi. Choose **Apply**, and then choose **Confirm**. A new Envoy proxy container is created and added to the task definition, and the settings to support the container are also created. The Envoy proxy container then pre-populates the App Mesh **Proxy Configuration** settings for the next step.

   b. For **Proxy Configuration**, verify all of the pre-populated values.

   c. For **Network Mode**, ensure that `awsvpc` is selected. To learn more about the `awsvpc` network mode, see Task Networking with the `awsvpc` Network Mode.

6. Choose **Create**.
7. Update your service with the updated task definition. For more information, see Updating a service.

JSON

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`

The Envoy proxy doesn't route traffic from processes that use this user ID. You can choose any user ID that you want for this property value, 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. Our examples use *`1337`* for historical purposes.

`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`*. The port that you specify must match the port configured on the virtual node listener.

`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. The metadata endpoint provides IAM roles for tasks credentials. You can add additional addresses.

`EgressIgnoredPorts`

You can add a comma separated list of ports. Envoy doesn't proxy traffic to these ports. Even if you list no ports, port 22 is ignored.

```
"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"
  },
  {
   "name": "EgressIgnoredPorts",
   "value": "22"
  }
```

```
 ]
}
```

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 shows an application container definition example with this dependency. All of the properties in the following example are required. Some of the property values are also required, but some are *`replaceable`*.

```
{
 "name": "appName",
 "image": "appImage",
 "portMappings": [{
  "containerPort": 9080,
  "hostPort": 9080,
  "protocol": "tcp"
 }],
 "essential": true,
 "dependsOn": [{
  "containerName": "envoy",
  "condition": "HEALTHY"
 }]
}
```

Envoy container definition

Your Amazon ECS task definitions must contain one of the following App Mesh Envoy container images:

- All supported Regions other than `me-south-1` and `ap-east-1`. You can replace *`us-west-2`* with any Region other than `me-south-1` and `ap-east-1`.

  ```
  840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
  ```

- `me-south-1` Region:

  ```
  772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
  ```

- `ap-east-1` Region:

  ```
  856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
  ```

You must use the App Mesh Envoy container image until the Envoy project team merges changes that support App Mesh. For additional details, see the GitHub roadmap issue.

All of the properties in the following example are required. Some of the property values are also required, but some are *`replaceable`*. The Envoy container definition must be marked as `essential`. The virtual node name for the Amazon ECS service must be set to the value of the `APPMESH_VIRTUAL_NODE_NAME` property. The value for the `user` setting must 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 the Envoy container is healthy and ready for the application containers to start.

The following code shows an Envoy container definition example.

```
{
 "name": "envoy",
 "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-
prod",
 "essential": true,
 "environment": [{
  "name": "APPMESH_VIRTUAL_NODE_NAME",
  "value": "mesh/apps/virtualNode/serviceB"
 }],
 "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 definitions

The following example Amazon ECS task definitions show how to merge the examples from above into a task definition for `taskB`. Examples are provided for creating tasks for both Amazon ECS launch types with or without using AWS X-Ray. Change the *`replaceable`* values, as appropriate, to create task definitions for the tasks named `taskBv2` and `taskA` from the scenario. 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. All of the properties in the following examples are required. Some of the property values are also required, but some are *`replaceable`*.

If you're running an Amazon ECS task as described in the Credentials section, then you need to add an existing task IAM role, to the examples.

Example JSON for Amazon ECS task definition - Fargate launch type

```
{

   "family" : "taskB",
   "memory" : "1024",
   "cpu" : "0.5 vCPU",
   "proxyConfiguration" : {
      "containerName" : "envoy",
      "properties" : [
         {
            "name" : "ProxyIngressPort",
            "value" : "15000"
         },
         {
            "name" : "AppPorts",
            "value" : "9080"
         },
         {
            "name" : "EgressIgnoredIPs",
            "value" : "169.254.170.2,169.254.169.254"
         },
         {
            "name": "EgressIgnoredPorts",
            "value": "22"
         },
```

```
            {
               "name" : "IgnoredUID",
               "value" : "1337"
            },
            {
               "name" : "ProxyEgressPort",
               "value" : "15001"
            }
         ],
         "type" : "APPMESH"
      },
      "containerDefinitions" : [
         {
            "name" : "appName",
            "image" : "appImage",
            "portMappings" : [
               {
                  "containerPort" : 9080,
                  "protocol" : "tcp"
               }
            ],
            "essential" : true,
            "dependsOn" : [
               {
                  "containerName" : "envoy",
                  "condition" : "HEALTHY"
               }
            ]
         },
         {
            "name" : "envoy",
            "image" : "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.12.3.0-prod",
            "essential" : true,
            "environment" : [
               {
                  "name" : "APPMESH_VIRTUAL_NODE_NAME",
                  "value" : "mesh/apps/virtualNode/serviceB"
               }
            ],
            "healthCheck" : {
               "command" : [
                  "CMD-SHELL",
                  "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
               ],
               "interval" : 5,
               "retries" : 3,
               "startPeriod" : 10,
               "timeout" : 2
            },
            "memory" : "500",
            "user" : "1337"
         }
      ],
      "requiresCompatibilities" : [ "FARGATE" ],
      "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
      "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
      "networkMode" : "awsvpc"
}
```

Example JSON for Amazon ECS task definition with AWS X-Ray - Fargate launch type

X-Ray allows you to collect data about requests that an application serves and provides tools that you can use to visualize traffic flow. Using the X-Ray driver for Envoy enables Envoy to report tracing information to X-Ray. You can enable X-Ray tracing using the Envoy configuration. Based on the

configuration, Envoy sends tracing data to the X-Ray daemon running as a sidecar container and the daemon forwards the traces to the X-Ray service. Once the traces are published to X-Ray, you can use the X-Ray console to visualize the service call graph and request trace details. The following JSON represents a task definition to enable X-Ray integration.

```
{


   "family" : "taskB",
   "memory" : "1024",
   "cpu" : "0.5 vCPU",
   "proxyConfiguration" : {
      "containerName" : "envoy",
      "properties" : [
         {
            "name" : "ProxyIngressPort",
            "value" : "15000"
         },
         {
            "name" : "AppPorts",
            "value" : "9080"
         },
         {
            "name" : "EgressIgnoredIPs",
            "value" : "169.254.170.2,169.254.169.254"
         },
         {
            "name": "EgressIgnoredPorts",
            "value": "22"
         },
         {
            "name" : "IgnoredUID",
            "value" : "1337"
         },
         {
            "name" : "ProxyEgressPort",
            "value" : "15001"
         }
      ],
      "type" : "APPMESH"
   },
   "containerDefinitions" : [
      {
         "name" : "appName",
         "image" : "appImage",
         "portMappings" : [
            {
               "containerPort" : 9080,
               "protocol" : "tcp"
            }
         ],
         "essential" : true,
         "dependsOn" : [
            {
               "containerName" : "envoy",
               "condition" : "HEALTHY"
            }
         ]
      },
      {

         "name" : "envoy",
         "image" : "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.12.3.0-prod",
         "essential" : true,
```

```
         "environment" : [
            {
               "name" : "APPMESH_VIRTUAL_NODE_NAME",
               "value" : "mesh/apps/virtualNode/serviceB"
            },
            {
               "name": "ENABLE_ENVOY_XRAY_TRACING",
               "value": "1"
            }
         ],
         "healthCheck" : {
            "command" : [
               "CMD-SHELL",
               "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
            ],
            "interval" : 5,
            "retries" : 3,
            "startPeriod" : 10,
            "timeout" : 2
         },
         "memory" : "500",
         "user" : "1337"
      },
      {
         "name" : "xray-daemon",
         "image" : "amazon/aws-xray-daemon",
         "user" : "1337",
         "essential" : true,
         "cpu" : "32",
         "memoryReservation" : "256",
         "portMappings" : [
            {
               "containerPort" : 2000,
               "protocol" : "udp"
            }
         ]
      }
   ],
   "requiresCompatibilities" : [ "FARGATE" ],
   "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
   "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
   "networkMode" : "awsvpc"
}
```

Example JSON for Amazon ECS task definition - EC2 launch type

```
{
  "family": "taskB",
  "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"
      },
      {
        "name": "EgressIgnoredPorts",
        "value": "22"
      }
    ]
  },
  "containerDefinitions": [
    {
      "name": "appName",
      "image": "appImage",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
      "name": "envoy",
      "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.12.3.0-prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/apps/virtualNode/serviceB"
        }
      ],
      "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"
    }
  ],
  "requiresCompatibilities" : [ "EC2" ],
  "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
  "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

Example JSON for Amazon ECS task definition with AWS X-Ray - EC2 launch type

X-Ray allows you to collect data about requests that an application serves and provides tools that you can use to visualize traffic flow. Using the X-Ray driver for Envoy enables Envoy to report tracing information to X-Ray. You can enable X-Ray tracing using the Envoy configuration. Based on the configuration, Envoy sends tracing data to the X-Ray daemon running as a sidecar container and the daemon forwards the traces to the X-Ray service. Once the traces are published to X-Ray, you can use the X-Ray console to visualize the service call graph and request trace details. The following JSON represents a task definition to enable X-Ray integration.

```
{
  "family": "taskB",
  "memory": "256",
   "cpu" : "1024",
  "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"
      },
      {
        "name": "EgressIgnoredPorts",
        "value": "22"
      }
    ]
  },
  "containerDefinitions": [
    {
      "name": "appName",
      "image": "appImage",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
```

```
      "name": "envoy",
      "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.12.3.0-prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/apps/virtualNode/serviceB"
        },
        {
         "name": "ENABLE_ENVOY_XRAY_TRACING",
         "value": "1"
        }
      ],
      "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"
    },
    {
      "name": "xray-daemon",
      "image": "amazon/aws-xray-daemon",
      "user": "1337",
      "essential": true,
      "cpu": 32,
      "memoryReservation": 256,
      "portMappings": [
        {
          "containerPort": 2000,
          "protocol": "udp"
        }
      ]
    }
  ],
  "requiresCompatibilities" : [ "EC2" ],
  "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
  "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

AWS Deep Learning Containers on Amazon ECS

AWS Deep Learning Containers provide a set of Docker images for training and serving models in TensorFlow and Apache MXNet on Amazon ECS. Deep Learning Containers enable optimized environments with TensorFlow, NVIDIA CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries. Container images for Deep Learning Containers are available in Amazon ECR to reference in Amazon ECS task definitions. You can use Deep Learning Containers along with Amazon Elastic Inference to lower your inference costs.

To get started using Deep Learning Containers without Elastic Inference on Amazon ECS, see Deep Learning Containers on Amazon ECS in the *AWS Deep Learning AMI Developer Guide*.

Deep Learning Containers with Elastic Inference on Amazon ECS

AWS Deep Learning Containers provide a set of Docker images for serving models in TensorFlow and Apache MXNet that take advantage of Amazon Elastic Inference accelerators. Amazon ECS provides task definition parameters to attach Elastic Inference accelerators to your containers. When you specify an Elastic Inference accelerator type in your task definition, Amazon ECS manages the lifecycle of, and configuration for, the accelerator. The Amazon ECS service-linked role is required when using this feature. For more information about Elastic Inference accelerators, see Amazon Elastic Inference Basics.

Important
Your Amazon ECS container instances require at least version `1.30.0` 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. 264).

To get started using Deep Learning Containers with Elastic Inference on Amazon ECS, see Deep Learning Containers with Elastic Inference on Amazon ECS in the *Amazon Elastic Inference Developer Guide*.

Tutorials for Amazon ECS

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

Topics

- Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters (p. 643)
- Tutorial: Creating a Cluster with a Fargate Task Using the AWS CLI (p. 645)
- Tutorial: Creating a Cluster with an EC2 Task Using the AWS CLI (p. 650)
- Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets (p. 657)
- Tutorial: Creating a Service Using Service Discovery (p. 662)
- Tutorial: Creating a Service Using a Blue/Green Deployment (p. 671)
- Tutorial: Listening for Amazon ECS CloudWatch Events (p. 680)
- Tutorial: Sending Amazon Simple Notification Service Alerts for Task Stopped Events (p. 682)
- Tutorial: Using Amazon EFS File Systems with Amazon ECS (p. 684)

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.

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 **Launch 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.

   Non-default subnets, such as those created by the VPC wizard, are not auto-assigned public IPv4 addresses. Instances launched in the public subnet must be assigned a public IPv4 address to communicate with the Amazon ECS service endpoint.

To modify your public subnet's IPv4 addressing behavior

1. In the left navigation pane, choose **Subnets**.
2. Select the public subnet for your VPC By default, the name created by the VPC wizard is **Public subnet**.
3. Choose **Actions**, **Modify auto-assign IP settings**.
4. Select the **Enable auto-assign public IPv4 address** check box, and then choose **Save**.

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. 38). 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. 376).

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. 645)
- Step 1: Create a Cluster (p. 646)
- Step 2: Register a Task Definition (p. 646)
- Step 3: List Task Definitions (p. 647)
- Step 4: Create a Service (p. 647)
- Step 5: List Services (p. 648)
- Step 6: Describe the Running Service (p. 648)
- Step 7: Clean Up (p. 650)

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 Permissions (p. 454) IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Docker Hub so your task must have internet access. To give your task a route to the internet, use one of the following options.
  - Use a private subnet with a NAT gateway that has an elastic IP address.
  - Use a public subnet and assign a public IP address to the task.

  For more information, see Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters.

Step 1: Create a Cluster

By default, your account receives a `default` cluster.

> **Note**
> The benefit of using the `default` cluster that is provided for you is that you don't have to specify the `--cluster` *`cluster_name`* option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster` *`cluster_name`* for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{
    "cluster": {
        "status": "ACTIVE",
        "statistics": [],
        "clusterName": "fargate-cluster",
        "registeredContainerInstancesCount": 0,
        "pendingTasksCount": 0,
        "runningTasksCount": 0,
        "activeServicesCount": 0,
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
    }
}
```

Step 2: Register a Task Definition

Before you can run a task on your ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that creates a PHP web app using the httpd container image hosted on Docker Hub. For more information about the available task definition parameters, see Amazon ECS Task Definitions (p. 73).

```
{
    "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> <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"
}
```

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** command returns a description of the task definition after it completes its registration.

Step 3: List Task Definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the `family` and `revision` values that you can use together when calling **run-task** or **start-task**.

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

Output:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:1"
    ]
}
```

Step 4: Create a Service

After you have registered a task for your account, you can create a service for the registered task in your cluster. For this example, you create a service with one instance of the `sample-fargate:1` task

definition running in your cluster. The task requires a route to the internet, so there are two ways you can achieve this. One way is to use a private subnet configured with a NAT gateway with an elastic IP address in a public subnet. Another way is to use a public subnet and assign a public IP address to your task. We provide both examples below.

Example using a private subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234]}"
```

Example using a public subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-
configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234],assignPublicIp=ENABLED}"
```

The **create-service** command returns a description of the task definition after it completes its registration.

Step 5: List Services

List the services for your cluster. You should see the service that you created in the previous section. You can take the service name or the full ARN that is returned from this command and use it to describe the service later.

```
aws ecs list-services --cluster fargate-cluster
```

Output:

```
{
    "serviceArns": [
        "arn:aws:ecs:region:aws_account_id:service/fargate-service"
    ]
}
```

Step 6: Describe the Running Service

Describe the service using the service name retrieved earlier to get more information about the task.

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

If successful, this will return a description of the service failures and services. For example, in services section, you will find information on deployments, such as the status of the tasks as running or pending. You may also find information on the task definition, the network configuration and time-stamped events. In the failures section, you will find information on failures, if any, associated with the call. For troubleshooting, see Service Event Messages. For more information about the service description, see Describe Services.

```
{
    "services": [
        {
            "status": "ACTIVE",
            "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-
fargate:1",
```

```
            "pendingCount": 2,
            "launchType": "FARGATE",
            "loadBalancers": [],
            "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/
ecs.amazonaws.com/AWSServiceRoleForECS",
            "placementConstraints": [],
            "createdAt": 1510811361.128,
            "desiredCount": 2,
            "networkConfiguration": {
                "awsvpcConfiguration": {
                    "subnets": [
                        "subnet-abcd1234"
                    ],
                    "securityGroups": [
                        "sg-abcd1234"
                    ],
                    "assignPublicIp": "DISABLED"
                }
            },
            "platformVersion": "LATEST",
            "serviceName": "fargate-service",
            "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
            "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
            "deploymentConfiguration": {
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "deployments": [
                {
                    "status": "PRIMARY",
                    "networkConfiguration": {
                        "awsvpcConfiguration": {
                            "subnets": [
                                "subnet-abcd1234"
                            ],
                            "securityGroups": [
                                "sg-abcd1234"
                            ],
                            "assignPublicIp": "DISABLED"
                        }
                    },
                    "pendingCount": 2,
                    "launchType": "FARGATE",
                    "createdAt": 1510811361.128,
                    "desiredCount": 2,
                    "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
sample-fargate:1",
                    "updatedAt": 1510811361.128,
                    "platformVersion": "0.0.1",
                    "id": "ecs-svc/9223370526043414679",
                    "runningCount": 0
                }
            ],
            "events": [
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
                    "id": "92b8443e-67fb-4886-880c-07e73383ea83",
                    "createdAt": 1510811841.408
                },
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
                    "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
                    "createdAt": 1510811601.938
                },
```

```
                {
                    "message": "(service fargate-service) has started 2 tasks: (task
 cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
                    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
                    "createdAt": 1510811364.691
                }
            ],
            "runningCount": 0,
            "placementStrategy": []
        }
    ],
    "failures": []
}
```

Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

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. 650)
- Step 1: Create a Cluster (p. 651)
- Step 2: Launch an Instance with the Amazon ECS AMI (p. 651)
- Step 3: List Container Instances (p. 651)
- Step 4: Describe your Container Instance (p. 652)
- Step 5: Register a Task Definition (p. 654)
- Step 6: List Task Definitions (p. 655)
- Step 7: Run a Task (p. 655)
- Step 8: List Tasks (p. 656)
- Step 9: Describe the Running Task (p. 656)

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 Permissions (p. 454) 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 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. 219) for more information.

Step 3: List Container Instances

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",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                },
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "ec2InstanceId": "instance_id",
            "agentConnected": true,
            "containerInstanceArn": "arn:aws:ecs:us-west-2:aws_account_id:container-
instance/container_instance_ID",
            "pendingTasksCount": 0,
            "remainingResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
```

```
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                },
                {
                    "name": "PORTS",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [
                        "22",
                        "2376",
                        "2375",
                        "51678"
                    ],
                    "type": "STRINGSET",
                    "integerValue": 0
                },
                {
                    "name": "PORTS_UDP",
                    "longValue": 0,
                    "doubleValue": 0.0,
                    "stringSetValue": [],
                    "type": "STRINGSET",
                    "integerValue": 0
                }
            ],
            "runningTasksCount": 0,
            "attributes": [
                {
                    "name": "com.amazonaws.ecs.capability.privileged-container"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
                },
                {
                    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
                },
                {
                    "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
                },
                {
                    "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
                }
            ],
            "versionInfo": {
                "agentVersion": "1.5.0",
                "agentHash": "b197edd",
                "dockerVersion": "DockerVersion: 1.7.1"
            }
        }
    ]
}
```


You can also find the Amazon EC2 instance ID that you can use to monitor the instance in the Amazon EC2 console or with the **aws ec2 describe-instances --instance-id *instance_id*** command.

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, see Amazon ECS Task Definitions (p. 73).

```
{
  "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/sleep360:1",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:3",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:4",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:5",
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
    ]
}
```

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.

```
aws ecs run-task --cluster default --task-definition sleep360:1 --count 1
```

Output:

```
{
    "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.

```
aws ecs list-tasks --cluster default
```

Output:

```
{
    "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": []
                }
            ]
        }
    ]
}
```

Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets

Amazon ECS enables you to inject sensitive data into your containers by storing your sensitive data in AWS Secrets Manager secrets and then referencing them in your container definition. For more information, see Specifying Sensitive Data (p. 163).

The following tutorial shows how to create an Secrets Manager secret, reference the secret in an Amazon ECS task definition, and then verify it worked by querying the environment variable inside a container showing the contents of the secret.

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in Setting Up with Amazon ECS (p. 7) have been completed.
- Your AWS user has the required IAM permissions to create the Secrets Manager and Amazon ECS resources described.

Step 1: Create an Secrets Manager Secret

You can use the Secrets Manager console to create a secret for your sensitive data. In this tutorial we will be creating a basic secret for storing a username and password to reference later in a container. For more information, see 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. For **Specify the key/value pairs to be stored in this secret**, choose the **Plaintext** tab and replace the existing text with the following text. The text value you specify here will be the environment variable value in your container at the end of the tutorial.

```
password_value
```

5. Choose **Next**.
6. For **Secret name**, type `username_value` and choose **Next**. The secret name value you specify here will be the environment variable name in your container at the end of the tutorial.
7. For **Configure automatic rotation**, leave **Disable automatic rotation** selected and choose **Next**.
8. Review these settings, and then choose **Store** to save everything you entered as a new secret in Secrets Manager.
9. Select the secret you just created and save the **Secret ARN** to reference in your task execution IAM policy and task definition in later steps.

Step 2: Update Your Task Execution IAM Role

In order for Amazon ECS to retrieve the sensitive data from your Secrets Manager secret, 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 Secrets Manager resources. If you have not already created your task execution IAM role, see Amazon ECS Task Execution IAM Role (p. 482).

The following steps assume you already have the task execution IAM role created and properly configured.

To update your task execution IAM role

Use the IAM console to update your task execution role with the required permissions.

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` and select it.
4. Choose **Permissions**, **Add inline policy**.
5. Choose the **JSON** tab and specify the following JSON text, ensuring that you specify the full ARN of the Secrets Manager secret you created in step 1.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:region:aws_account_id:secret:username_value-
u9bH6K"
            ]
        }
    ]
}
```

6. Choose **Review policy**. For **Name** specify `ECSSecretsTutorial`, then choose **Create policy**.

Step 3: Create an Amazon ECS Task Definition

You can use the Amazon ECS console to create a task definition that references a Secrets Manager secret.

To create a task definition that specifies a secret

Use the IAM console to update your task execution role with the required permissions.

1. Open the 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 **EC2** and choose **Next step**.
4. Choose **Configure via JSON** and enter the following task definition JSON text, ensuring that you specify the full ARN of the Secrets Manager secret you created in step 1 and the task execution IAM role you updated in step 2. Choose **Save**.

   Important
   The value for the secret name in the task definition must match the name you specified for the secret name when the secret was created.

```
{
    "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
 App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
 <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
 ECS.</p> </div></body></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
            ],
            "cpu": 10,
            "secrets": [
                {
                    "valueFrom":
 "arn:aws:secretsmanager:region:aws_account_id:secret:username_value-u9bH6K",
                    "name": "username_value"
                }
            ],
            "memory": 300,
            "image": "httpd:2.4",
            "essential": true,
            "name": "ecs-secrets-container"
        }
    ],
    "family": "ecs-secrets-tutorial"
}
```

5. Review the settings and then choose **Create**.

Step 4: Create an Amazon ECS Cluster

You can use the Amazon ECS console to create a cluster containing a container instance to run the task on. If you have an existing cluster with at least one container instance registered to it with the available resources to run one instance of the task definition created for this tutorial you can skip to the next step.

For this tutorial we will be creating a cluster with one `t2.micro` container instance using the Amazon ECS-optimized Amazon Linux 2 AMI.

To create a cluster

Use the Amazon ECS console to create a cluster and register one container instance to it.

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, select the Region that contains both the Secrets Manager secret and the Amazon ECS task definition you created.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. For **EC2 instance type**, choose **t2.micro**.
6. For **Key pair**, choose a key pair to add to the container instance.

   Important
   A key pair is required to complete the tutorial, so if you do not already have a key pair created follow the link to the EC2 console to create one.
7. Leave all other fields at their default values and choose **Create**.

Step 5: Run an Amazon ECS Task

You can use the Amazon ECS console to run a task using the task definition you created. For this tutorial we will be running a task using the EC2 launch type, using the cluster we created in the previous step.

To run a task

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose **Task Definitions** and select the **ecs-secrets-tutorial** task definition we created.
3. Select the latest revision of the task definition and then choose **Actions**, **Run Task**.
4. For **Launch Type**, choose **EC2**.
5. For **Cluster**, choose the **ecs-secrets-tutorial** cluster we created in the previous step.
6. For **Task tagging configuration**, deselect **Enable ECS managed tags**. They are unnecessary for the purposes of this tutorial.
7. 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. 690) in the troubleshooting section.

Step 6: Verify

You can verify all of the steps were completed successfully and the environment variable was created properly in your container using the following steps.

To verify that the environment variable was created

1. Find the public IP or DNS address for your container instance.
   a. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
   b. Select the `ecs-secrets-tutorial` 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. 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*.

3. List the containers running on the instance. Note the container ID for `ecs-secrets-tutorial` container.

```
docker ps
```

4. Connect to the `ecs-secrets-tutorial` container using the container ID from the output of the previous step.

```
docker exec -it container_ID /bin/bash
```

5. Use the `echo` command to print the value of the environment variable.

```
echo $username_value
```

If the tutorial was successful, you should see the following output:

```
password_value
```

Note
Alternatively, you can list all environment variables in your container using the `env` (or `printenv`) command.

Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

To clean up the resources

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Select the **ecs-secrets-tutorial cluster** you created.
3. On the **Cluster** page, choose **Delete Cluster**.
4. Enter the delete cluster confirmation phrase and choose **Delete**. This will take several minutes but will clean up all of the Amazon ECS cluster resources.
5. Open the IAM console at https://console.aws.amazon.com/iam/.
6. In the navigation pane, choose **Roles**.
7. Search the list of roles for `ecsTaskExecutionRole` and select it.
8. Choose **Permissions**, then choose the **X** next to `ECSSecretsTutorial`. Choose **Remove** to confirm the removal of the inline policy.

9. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
10. Select the **username_value** secret you created and choose **Actions**, **Delete secret**.

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. 352).

The following tutorial shows how to create an ECS service containing a Fargate task that uses service discovery with the AWS CLI.

For a list of Regions that support service discovery, see Service Discovery (p. 403).

Fargate tasks are only supported in the following Regions:

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1 (`aps1-az1` & `aps1-az3` only)
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
Middle East (Bahrain)	me-south-1
AWS GovCloud (US-East)	us-gov-east-1

Region Name	Region
AWS GovCloud (US-West)	us-gov-west-1

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 Permissions (p. 454) 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": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-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 h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-
je6chs2e
```

   Output:

```
{
    "Operation": {
        "Id": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e",
        "Type": "CREATE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1519777852.502,
        "UpdateDate": 1519777856.086,
        "Targets": {
            "NAMESPACE": "ns-uejictsjen2i4eeg"
        }
    }
}
```

3. Using the `NAMESPACE` ID from the 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-utcrh6wavdkggqtk",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-
utcrh6wavdkggqtk",
        "Name": "myapplication",
        "DnsConfig": {
            "NamespaceId": "ns-uejictsjen2i4eeg",
            "DnsRecords": [
                {
                    "Type": "A",
                    "TTL": 300
                }
            ]
        },
        "HealthCheckCustomConfig": {
            "FailureThreshold": 1
        },
        "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"
    }
}
```

Step 2: Create the Amazon ECS Resources

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></html>' >  /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
                ]
            }
        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
        "memory": "512"
}
```

Then, register the task definition using the `fargate-task.json` file that you created:

```
aws ecs register-task-definition --cli-input-json file://fargate-task.json --region us-
east-1
```

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.

```
{
    "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:

```
aws ecs create-service --cli-input-json file://ecs-service-discovery.json --region us-east-1
```

Output:

```
{
    "service": {
        "serviceArn": "arn:aws:ecs:region:aws_account_id:service/ecs-service-
discovery",
        "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"
            }
        ],
        "status": "ACTIVE",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "FARGATE",
        "platformVersion": "LATEST",
        "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/tutorial-
task-def:1",
        "deploymentConfiguration": {
            "maximumPercent": 200,
            "minimumHealthyPercent": 100
        },
        "deployments": [
            {
                "id": "ecs-svc/9223370516993140842",
                "status": "PRIMARY",
                "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
tutorial-task-def:1",
                "desiredCount": 1,
                "pendingCount": 0,
                "runningCount": 0,
                "createdAt": 1519861634.965,
                "updatedAt": 1519861634.965,
                "launchType": "FARGATE",
                "platformVersion": "1.1.0",
                "networkConfiguration": {
                    "awsvpcConfiguration": {
                        "subnets": [
                            "subnet-abcd1234"
                        ],
                        "securityGroups": [
                            "sg-abcd1234"
                        ],
                        "assignPublicIp": "ENABLED"
```

```
                    }
                }
            }
        ],
        "roleArn": "arn:aws:iam::aws_account_id:role/ECSServiceLinkedRole",
        "events": [],
        "createdAt": 1519861634.965,
        "placementConstraints": [],
        "placementStrategy": [],
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "subnets": [
                    "subnet-abcd1234"
                ],
                "securityGroups": [
                    "sg-abcd1234"
                ],
                "assignPublicIp": "ENABLED"
            }
        }
    }
}
```

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-utcrh6wavdkggqtk --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",
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-
uejictsjen2i4eeg",
        "Name": "tutorial",
        "Type": "DNS_PRIVATE",
        "Properties": {
            "DnsProperties": {
                "HostedZoneId": "Z35JQ4ZFDRYPLV"
            }
        },
        "CreateDate": 1519777852.502,
        "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
    }
}
```

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 Z35JQ4ZFDRYPLV --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"
                }
            ]
        },
        {
            "Name": "myapplication.tutorial.",
            "Type": "A",
            "SetIdentifier": "16becc26-8558-4af1-9fbd-f81be062a266",
            "MultiValueAnswer": true,
            "TTL": 300,
            "ResourceRecords": [
                {
                    "Value": "172.31.87.2"
                }
            ]
        }
    ]
}
```

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-utcrh6wavdkggqtk --instance-id 16becc26-8558-4af1-9fbd-f81be062a266 --region us-east-1
```

   Output:

```
{
    "OperationId": "xhu73bsertlyffhm3faqi7kumsmx274n-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 xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv --region us-east-1
```

   Output:

```
{
    "Operation": {
        "Id": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv",
        "Type": "DEREGISTER_INSTANCE",
        "Status": "SUCCESS",
        "CreateDate": 1525984073.707,
        "UpdateDate": 1525984076.426,
        "Targets": {
            "INSTANCE": "16becc26-8558-4af1-9fbd-f81be062a266",
            "ROUTE_53_CHANGE_ID": "C5NSRG1J4I1FH",
            "SERVICE": "srv-utcrh6wavdkggqtk"
        }
    }
}
```

3. Delete the service discovery service:

```
aws servicediscovery delete-service --id srv-utcrh6wavdkggqtk --region us-east-1
```

4. Delete the service discovery namespace:

```
aws servicediscovery delete-namespace --id ns-uejictsjen2i4eeg --region us-east-1
```

   Output:

```
{
    "OperationId": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj"
}
```

5. Using the `OperationId` from the previous output, verify that the service discovery namespace was deleted successfully:

```
aws servicediscovery get-operation --operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj --region us-east-1
```

Output:

```
{
    "Operation": {
        "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
        "Type": "DELETE_NAMESPACE",
        "Status": "SUCCESS",
        "CreateDate": 1525984602.211,
        "UpdateDate": 1525984602.558,
        "Targets": {
            "NAMESPACE": "ns-rymlehshst7hhukh",
            "ROUTE_53_CHANGE_ID": "CJP2A2M86XW3O"
        }
    }
}
```

6. Update the Amazon ECS service so that the desired count is `0`, which allows you to delete it:

```
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:

```
aws ecs delete-service --cluster tutorial --service ecs-service-discovery --region us-east-1
```

8. Delete the Amazon ECS cluster:

```
aws ecs delete-cluster --cluster tutorial --region us-east-1
```

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. 352).

The following tutorial shows how to create an Amazon ECS service containing a Fargate task that uses the blue/green deployment type with the AWS CLI.

Note
Support for performing a blue/green deployment has been added for AWS CloudFormation. For more information, see Perform Amazon ECS blue/green deployments through CodeDeploy using AWS CloudFormation in the *AWS CloudFormation User Guide*.

Prerequisites

This tutorial assumes that you have completed the following prerequisites:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading the AWS CLI, see Installing the AWS Command Line Interface.

- The steps in 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 Permissions (p. 454) 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.
- The Amazon ECS CodeDeploy IAM role is created. For more information, see Amazon ECS CodeDeploy IAM Role (p. 493).

Step 1: Create an Application Load Balancer

Amazon ECS services using the blue/green deployment type require the use of either an Application Load Balancer or a Network Load Balancer. This tutorial uses an Application Load Balancer.

To create an Application Load Balancer

1. Use the create-load-balancer command to create an Application Load Balancer. Specify two subnets that aren't from the same Availability Zone as well as a security group.

```
aws elbv2 create-load-balancer \
     --name bluegreen-alb \
     --subnets subnet-abcd1234 subnet-abcd5678 \
     --security-groups sg-abcd1234 \
     --region us-east-1
```

   The output includes the Amazon Resource Name (ARN) of the load balancer, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642
```

2. Use the create-target-group command to create a target group. This target group will route traffic to the original task set in your service.

```
aws elbv2 create-target-group \
     --name bluegreentarget1 \
     --protocol HTTP \
     --port 80 \
     --target-type ip \
     --vpc-id vpc-abcd1234 \
     --region us-east-1
```

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

```
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4
```

3. Use the create-listener command to create a load balancer listener with a default rule that forwards requests to the target group.

```
aws elbv2 create-listener \
     --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
     --protocol HTTP \
     --port 80 \
```

```
    --default-actions
 Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4 \
    --region us-east-1
```

The output includes the ARN of the listener, with the following format:

```
arn:aws:elasticloadbalancing:region:aws_account_id:listener/app/bluegreen-alb/
e5ba62739c16e642/665750bec1b03bd4
```

Step 2: Create an Amazon ECS Cluster

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

```
aws ecs create-cluster \
    --cluster-name tutorial-bluegreen-cluster \
    --region us-east-1
```

The output includes the ARN of the cluster, with the following format:

```
arn:aws:ecs:region:aws_account_id:cluster/tutorial-bluegreen-cluster
```

Step 3: Register a Task Definition

Use the register-task-definition command to register a task definition that is compatible with Fargate. It requires the use of the `awsvpc` network mode. The following is the example task definition used for this tutorial.

First, create a file named `fargate-task.json` with the following contents. Ensure that you use the ARN for your task execution role. For more information, see Amazon ECS Task Execution IAM Role (p. 482).

```
{
    "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.

```
{
    "cluster": "tutorial-bluegreen-cluster",
    "serviceName": "service-bluegreen",
    "taskDefinition": "tutorial-task-def",
    "loadBalancers": [
        {
            "targetGroupArn":
 "arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4",
            "containerName": "sample-app",
            "containerPort": 80
        }
    ],
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "platformVersion": "LATEST",
    "networkConfiguration": {
       "awsvpcConfiguration": {
          "assignPublicIp": "ENABLED",
          "securityGroups": [ "sg-abcd1234" ],
          "subnets": [ "subnet-abcd1234", "subnet-abcd5678" ]
       }
    },
    "desiredCount": 1
}
```

Then create your service using the `service-bluegreen.json` file that you created.

```
aws ecs create-service \
     --cli-input-json file://service-bluegreen.json \
     --region us-east-1
```

The output includes the ARN of the service, with the following format:

```
arn:aws:ecs:region:aws_account_id:service/service-bluegreen
```

Step 5: Create the AWS CodeDeploy Resources

Use the following steps to create your CodeDeploy application, the Application Load Balancer target group for the CodeDeploy deployment group, and the CodeDeploy deployment group.

To create CodeDeploy resources

1. Use the create-application command to create an CodeDeploy application. Specify the `ECS` compute platform.

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

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

   ```
   {
       "applicationId": "b8e9c1ef-3048-424e-9174-885d7dc9dc11"
   }
   ```

2. Use the create-target-group command to create a second Application Load Balancer target group, which will be used when creating your CodeDeploy deployment group.

   ```
   aws elbv2 create-target-group \
        --name bluegreentarget2 \
        --protocol HTTP \
        --port 80 \
        --target-type ip \
        --vpc-id "vpc-0b6dd82c67d8012a1" \
        --region us-east-1
   ```

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

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

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

   First, create a file named `tutorial-deployment-group.json` with the following contents. This example uses the resource that you created. For the `serviceRoleArn`, specify the ARN of your Amazon ECS CodeDeploy IAM role. For more information, see Amazon ECS CodeDeploy IAM Role (p. 493).

   ```
   {
      "applicationName": "tutorial-bluegreen-app",
      "autoRollbackConfiguration": {
         "enabled": true,
         "events": [ "DEPLOYMENT_FAILURE" ]
      },
      "blueGreenDeploymentConfiguration": {
         "deploymentReadyOption": {
            "actionOnTimeout": "CONTINUE_DEPLOYMENT",
            "waitTimeInMinutes": 0
         },
         "terminateBlueInstancesOnDeploymentSuccess": {
   ```

```
         "action": "TERMINATE",
         "terminationWaitTimeInMinutes": 5
      }
   },
   "deploymentGroupName": "tutorial-bluegreen-dg",
   "deploymentStyle": {
      "deploymentOption": "WITH_TRAFFIC_CONTROL",
      "deploymentType": "BLUE_GREEN"
   },
   "loadBalancerInfo": {
      "targetGroupPairInfoList": [
        {
          "targetGroups": [
             {
                 "name": "bluegreentarget1"
             },
             {
                 "name": "bluegreentarget2"
             }
          ],
          "prodTrafficRoute": {
              "listenerArns": [
                  "arn:aws:elasticloadbalancing:region:aws_account_id:listener/
app/bluegreen-alb/e5ba62739c16e642/665750bec1b03bd4"
              ]
          }
        }
      ]
   },
   "serviceRoleArn": "arn:aws:iam::aws_account_id:role/ecsCodeDeployRole",
   "ecsServices": [
       {
           "serviceName": "service-bluegreen",
           "clusterName": "tutorial-bluegreen-cluster"
       }
   ]
}
```

Then create the CodeDeploy deployment group.

```
aws deploy create-deployment-group \
     --cli-input-json file://tutorial-deployment-group.json \
     --region us-east-1
```

The output includes the deployment group ID, with the following format:

```
{
    "deploymentGroupId": "6fd9bdc6-dc51-4af5-ba5a-0a4a72431c88"
}
```

Step 6: Create and Monitor an CodeDeploy Deployment

Use the following steps to create and upload an application specification file (AppSpec file) and an CodeDeploy deployment.

To create and monitor an CodeDeploy deployment

1. Create and upload an AppSpec file using the following steps.

a. Create a file named `appspec.yaml` with the contents of the CodeDeploy deployment group. This example uses the resources that you created earlier in the tutorial.

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "arn:aws:ecs:region:aws_account_id:task-definition/first-run-task-definition:7"
        LoadBalancerInfo:
          ContainerName: "sample-app"
          ContainerPort: 80
        PlatformVersion: "LATEST"
```

b. Use the s3 mb command to create an Amazon S3 bucket for the AppSpec file.

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

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

```
aws s3 cp ./appspec.yaml s3://tutorial-bluegreen-bucket/appspec.yaml
```

2. Create the CodeDeploy deployment using the following steps.

a. Create a file named `create-deployment.json` with the contents of the CodeDeploy deployment. This example uses the resources that you created earlier in the tutorial.

```
{
    "applicationName": "tutorial-bluegreen-app",
    "deploymentGroupName": "tutorial-bluegreen-dg",
    "revision": {
        "revisionType": "S3",
        "s3Location": {
            "bucket": "tutorial-bluegreen-bucket",
            "key": "appspec.yaml",
            "bundleType": "YAML"
        }
    }
}
```

b. Use the create-deployment command to create the deployment.

```
aws deploy create-deployment \
     --cli-input-json file://create-deployment.json \
     --region us-east-1
```

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

```
{
    "deploymentId": "d-RPCR1U3TW"
}
```

c. Use the get-deployment-target command to get the details of the deployment, specifying the `deploymentId` from the previous output.

```
aws deploy get-deployment-target \
     --deployment-id "d-IMJU3A8TW" \
     --target-id tutorial-bluegreen-cluster:service-bluegreen \
```

```
    --region us-east-1
```

Continue to retrieve the deployment details until the status is `Succeeded`, as shown in the following output.

```
{
    "deploymentTarget": {
        "deploymentTargetType": "ECSTarget",
        "ecsTarget": {
            "deploymentId": "d-RPCR1U3TW",
            "targetId": "tutorial-bluegreen-cluster:service-bluegreen",
            "targetArn": "arn:aws:ecs:region:aws_account_id:service/service-
bluegreen",
            "lastUpdatedAt": 1543431490.226,
            "lifecycleEvents": [
                {
                    "lifecycleEventName": "BeforeInstall",
                    "startTime": 1543431361.022,
                    "endTime": 1543431361.433,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "Install",
                    "startTime": 1543431361.678,
                    "endTime": 1543431485.275,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AfterInstall",
                    "startTime": 1543431485.52,
                    "endTime": 1543431486.033,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "BeforeAllowTraffic",
                    "startTime": 1543431486.838,
                    "endTime": 1543431487.483,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AllowTraffic",
                    "startTime": 1543431487.748,
                    "endTime": 1543431488.488,
                    "status": "Succeeded"
                },
                {
                    "lifecycleEventName": "AfterAllowTraffic",
                    "startTime": 1543431489.152,
                    "endTime": 1543431489.885,
                    "status": "Succeeded"
                }
            ],
            "status": "Succeeded",
            "taskSetsInfo": [
                {
                    "identifer": "ecs-svc/9223370493425779968",
                    "desiredCount": 1,
                    "pendingCount": 0,
                    "runningCount": 1,
                    "status": "ACTIVE",
                    "trafficWeight": 0.0,
                    "targetGroup": {
                        "name": "bluegreentarget1"
                    }
```

```
                },
                {
                    "identifer": "ecs-svc/9223370493423413672",
                    "desiredCount": 1,
                    "pendingCount": 0,
                    "runningCount": 1,
                    "status": "PRIMARY",
                    "trafficWeight": 100.0,
                    "targetGroup": {
                        "name": "bluegreentarget2"
                    }
                }
            ]
        }
    }
}
```

Step 7: Clean Up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using.

Cleaning up the tutorial resources

1. Use the delete-deployment-group command to delete the CodeDeploy deployment group.

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

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

```
aws deploy delete-application \
     --application-name tutorial-bluegreen-app \
     --region us-east-1
```

3. Use the delete-service command to delete the Amazon ECS service. Using the `--force` flag allows you to delete a service even if it has not been scaled down to zero tasks.

```
aws ecs delete-service \
     --service arn:aws:ecs:region:aws_account_id:service/service-bluegreen \
     --force \
     --region us-east-1
```

4. Use the delete-cluster command to delete the Amazon ECS cluster.

```
aws ecs delete-cluster \
     --cluster tutorial-bluegreen-cluster \
     --region us-east-1
```

5. Use the s3 rm command to delete the AppSpec file from the Amazon S3 bucket.

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

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

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

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

```
aws elbv2 delete-load-balancer \
     --load-balancer-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
     --region us-east-1
```

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

```
aws elbv2 delete-target-group \
     --target-group-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4 \
     --region us-east-1
```

```
aws elbv2 delete-target-group \
     --target-group-arn
 arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/708d384187a3cfdc \
     --region us-east-1
```

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 Creating a Cluster (p. 38) 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:

```
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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation pane, choose **Events**, **Rules**, **Create rule**.
3. For **Event Source**, choose **ECS** as the event source. By default, the rule applies to all Amazon ECS events for all of your Amazon ECS groups. Alternatively, you can select specific events or a specific Amazon ECS group.
4. For **Targets**, choose **Add target**, for **Target type**, choose **Lambda function**, and then select your Lambda function.
5. Choose **Configure details**.
6. For **Rule definition**, type a name and description for your rule and choose **Create rule**.

Step 3: 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**.
3. On the **Cluster : default** screen, choose **Tasks**, **Run new Task**.
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 Creating a Cluster (p. 38) 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/v3/home.
2. Choose **Topics**, **Create topic**.
3. On the **Create topic** screen, for **Name**, enter **TaskStoppedAlert** and choose **Create topic**.
4. On the **TaskStoppedAlert** details screen, choose **Create subscription**.
5. On the **Create subscription** screen, 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

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation pane, choose **Events**, **Rules**, **Create rule**.
3. For Event Source, choose **Event Pattern**, select **Custom event pattern** and then 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*.

4. For **Targets**, choose **Add target**. For **Target type**, choose **SNS topic**, and then choose **TaskStoppedAlert**.
5. Choose **Configure details**.
6. For **Rule definition**, type a name and description for your rule and then choose **Create rule**.

Step 3: Test Your Rule

Verify that the rule is working by running a task that exits shortly after it starts. If your event rule is configured correctly, you receive an email message within a few minutes with the event text. If you have an existing task definition that can satisfy the rule requirements, run a task using it. If you do not, the following steps will walk you through registering a Fargate task definition and running it that will.

To test the rule

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. Choose **Task Definitions**, **Create new Task Definition**.
3. For Select launch type compatibility, choose **FARGATE**, **Next step**.
4. Choose **Configure via JSON**, copy and paste the following task definition JSON into the field and choose **Save**.

```
{
   "containerDefinitions":[
      {
         "command":[
            "sh",
            "-c",
            "sleep 5"
         ],
         "essential":true,
         "image":"amazonlinux:2",
         "name":"test-sleep"
      }
   ],
   "cpu":"256",
   "executionRoleArn":"arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
   "family":"fargate-task-definition",
   "memory":"512",
   "networkMode":"awsvpc",
   "requiresCompatibilities":[
      "FARGATE"
   ]
}
```

5. Choose **Create**, **View task definition**.
6. For **Actions**, choose **Run Task**.
7. For Launch type, choose **FARGATE**. For **VPC and security groups**, choose a VPC and Subnets for the task to use and then choose **Run Task**.

8. For **Container name**, type **Wordpress**, for **Image**, type **wordpress**, and for **Maximum memory (MB)**, type **128**.
9. 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**.
10. Check your email to confirm that you have received an email alert for the stopped notification.

Tutorial: Using Amazon EFS File Systems with Amazon ECS

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic, 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 access file system data across your fleet of Amazon ECS tasks. That way, your tasks have access to the same persistent storage, no matter the infrastructure or container instance on which they land. When you reference your Amazon EFS file system and container mount point in your Amazon ECS task definition, Amazon ECS takes care of mounting the file system in your container. The following sections help you get started using Amazon EFS with Amazon ECS.

This feature is supported by tasks that use both the EC2 and Fargate launch types, however this tutorial will use an Amazon ECS task that uses the EC2 launch type. This tutorial is also meant to be followed step by step, however if you already have some of these resources created on your account then you may be able to skip some steps.

> **Note**
> Amazon EFS may not be available in all Regions. For more information about which Regions support Amazon EFS, see Amazon Elastic File System Endpoints and Quotas in the *AWS General Reference*.

Step 1: Create an Amazon ECS cluster

Use the following steps to create an Amazon ECS cluster. When you use the AWS Management Console to create a non-empty cluster, Amazon ECS creates an AWS CloudFormation stack along with Auto Scaling resources.

To create an Amazon ECS cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. For **Select cluster compatibility**, choose **EC2 Linux + Networking** and then choose **Next step**.
6. For **Cluster name**, enter `EFS-tutorial` for the cluster name.
7. For **Provisioning model**, choose **On-Demand Instance**.
8. For **EC2 instance type**, choose `t2.micro`.
9. For **Number of instances**, enter `1`.
10. For **EC2 AMI Id**, choose the Amazon Linux 2 Amazon ECS-optimized AMI.
11. For **EBS storage (GiB)**, leave the default setting.
12. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for SSH access. This is required as you will connect to the instance later.

13. 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 steps below.

    Important
    Record the VPC and security group IDs you use for your cluster as you will need to create the Amazon EFS file system in the same VPC.

    a. For **VPC**, create a new VPC, or select an existing VPC.

    b. (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*.

    c. For **Subnets**, select the subnets to use for your VPC. If you choose to create a new VPC, you can keep the default settings or you can modify them to meet your needs. If you choose 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
    The IAM role you use must have the `AmazonEC2ContainerServiceforEC2Role` managed policy attached to it, otherwise you will receive an error during cluster creation. 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. 486).

    f. For **CloudWatch Container Insights**, deselect **Enable Container Insights** as this feature won't be needed for this tutorial.

    g. Choose **Create**.

Step 2: Create a security group for the Amazon EFS file system

In this step, you create a security group for your Amazon EFS file system that allows inbound access from your container instances. This security group will contain an inbound rule that references the security group you created, or referenced, for your cluster in the previous step.

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-dc025fa2`.
4. For **Description**, enter a description for your security group.

5. For **VPC**, choose the VPC that you identified earlier for your cluster.
6. For **Inbound rules**, choose **Add rule**.
7. For **Type**, choose **NFS**.
8. For **Source**, choose **Custom** and then enter the security group ID that you identified earlier for your cluster.
9. Choose **Create security group**.

Step 3: Create an Amazon EFS file system

In this step, you create an Amazon EFS file system.

To create an Amazon EFS file system for Amazon ECS tasks.

1. Open the Amazon Elastic File System console at https://console.aws.amazon.com/efs/.
2. Choose **Create file system**.
3. On the **Configure network access** page, choose the VPC that your container instances are hosted in. By default, each subnet in the specified VPC receives a mount target that uses the default security group for that VPC.

   Important
   Your Amazon EFS file system, your Amazon ECS cluster, container instances and tasks must be in the same VPC.
4. Under **Create mount targets**, for **Security groups**, add the security group that you created in step 2. Choose **Next Step**.
5. On the **Configure file system settings** page, configure optional settings and then choose **Next Step** to proceed.

   a. (Optional) Add tags for your file system. For example, you could specify a unique name for the file system by entering that name in the **Value** column next to the **Name** key.
   b. (Optional) Enable lifecycle management to save money on infrequently accessed storage. For more information, see EFS Lifecycle Management in the *Amazon Elastic File System User Guide*.
   c. Choose a throughput mode for your file system. The **Bursting** mode is the default, and it is recommended for most file systems.
   d. Choose a performance mode for your file system. The **General Purpose** mode is the default, and it is recommended for most file systems.
   e. (Optional) Enable encryption. Select the check box to enable encryption of your Amazon EFS file system at rest.
6. On the **Configure client access** page, choose **Next Step**.
7. Review your file system options and choose **Create File System** to complete the process.
8. From the file systems details screen, record the **File system ID**. In the next step, you will reference this value in your Amazon ECS task definition.

Step 4: Add content to the Amazon EFS file system

In this step, you mount the Amazon EFS file system to an Amazon EC2 instance and add content to it. This is for testing purposes in this tutorial, to illustrate the persistent nature of the data. When using this feature you would normally have your application or another method of writing data to your Amazon EFS file system.

To create an Amazon EC2 instance and mount the Amazon EFS file system

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. Choose **Launch Instance**.
3. On the **Choose an Amazon Machine Image** page, select the latest **Amazon Linux 2 AMI (HVM)** AMI.
4. On the **Choose an Instance Type** page, keep the default instance type, `t2.micro` and choose **Next: Configure Instance Details**.
5. On the **Configure Instance Details** page, do the following:
   a. For **Network**, select the VPC that you specified for your Amazon EFS file system and Amazon ECS cluster.
   b. For **Auto-assign Public IP**, choose **Enable**. Otherwise, your instances do not get public IP addresses or public DNS names.
   c. For **File systems**, select your Amazon EFS file system. You can optionally change the mount location or leave the default value.
   d. Under **Advanced Details**, ensure that the user data script is populated automatically with the Amazon EFS file system mounting steps.
6. Advance to step 5 of the instance wizard by choosing **Next: Add Storage**, **Next: Add Tags**, and **Next: Configure Security Group**.
7. On the **Configure Security Group** page, choose **Select an existing security group** and select the security group that you created in step 1, and then choose **Review and Launch**.
8. On the **Review Instance Launch** page, choose **Launch**.
9. On the **Select an existing key pair or create a new key pair** dialog box, select **Choose an existing key pair** and choose your key pair. Select the acknowledgment check box, and choose **Launch Instances**.
10. On the **Launch Status** page, choose **View Instances** to see the status of your instances. Initially, their status is `pending`. After the status changes to `running`, your instances are ready for use.

Now, you connect to the Amazon EC2 instance and add content to the Amazon EFS file system.

To connect to the Amazon EC2 instance and add content to the Amazon EFS file system

1. SSH to the Amazon EC2 instance you created. For more information, see Connect to Your Linux Instance in the *Amazon EC2 User Guide for Linux Instances*.
2. From the terminal window for each instance, run the **df -T** command to verify that the Amazon EFS file system is mounted. In the following output, we have highlighted the Amazon EFS file system mount.

```
$ df -T
Filesystem     Type            1K-blocks    Used        Available Use% Mounted on
devtmpfs       devtmpfs           485468       0           485468   0% /dev
tmpfs          tmpfs              503480       0           503480   0% /dev/shm
tmpfs          tmpfs              503480     424           503056   1% /run
tmpfs          tmpfs              503480       0           503480   0% /sys/fs/cgroup
/dev/xvda1     xfs               8376300 1310952          7065348  16% /
127.0.0.1:/    nfs4     9007199254739968       0 9007199254739968   0% /mnt/efs/fs1
tmpfs          tmpfs              100700       0           100700   0% /run/user/1000
```

3. Navigate to the directory that the Amazon EFS file system is mounted at. In the example above, that is `/mnt/efs/fs1`.
4. Create a file named `index.html` with the following content:

```
<html>
    <body>
        <h1>It Works!</h1>
        <p>You are using an Amazon EFS file system for persistent container storage.</
p>
    </body>
```

```
</html>
```

Step 5: Create a task definition

The following task definition creates a data volume named `efs-html`. The `nginx` container mounts the host data volume at the NGINX root, `/usr/share/nginx/html`.

To create a new task definition

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, choose **EC2**, **Next step**.
4. Choose **Configure via JSON**, copy and paste the following JSON text, replacing the `fileSystemId` with the ID of your Amazon EFS file system.

```
{
    "containerDefinitions": [
        {
            "memory": 128,
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "mountPoints": [
                {
                    "containerPath": "/usr/share/nginx/html",
                    "sourceVolume": "efs-html"
                }
            ],
            "name": "nginx",
            "image": "nginx"
        }
    ],
    "volumes": [
        {
            "name": "efs-html",
            "efsVolumeConfiguration": {
                "fileSystemId": "fs-1324abcd",
                "transitEncryption": "ENABLED"
            }
        }
    ],
    "family": "efs-tutorial"
}
```

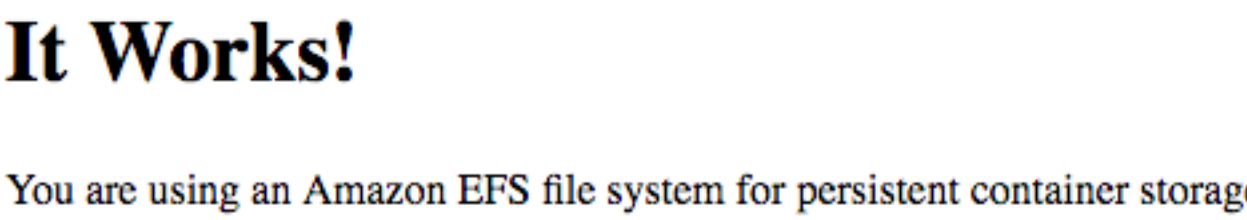
5. Choose **Save**, **Create**.

Step 6: Run a task and view the results

Now that your Amazon EFS file system is created and there is web content for the NGINX container to serve, you can run a task using the task definition that you created. The NGINX web server serves your simple HTML page. If you update the content in your Amazon EFS file system, those changes are propagated to any containers that have also mounted that file system.

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 created in step 1 earlier.
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. 316).
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:



Note
If you do not see the message, make sure that the security group for your container instance allows inbound network traffic on port 80.

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

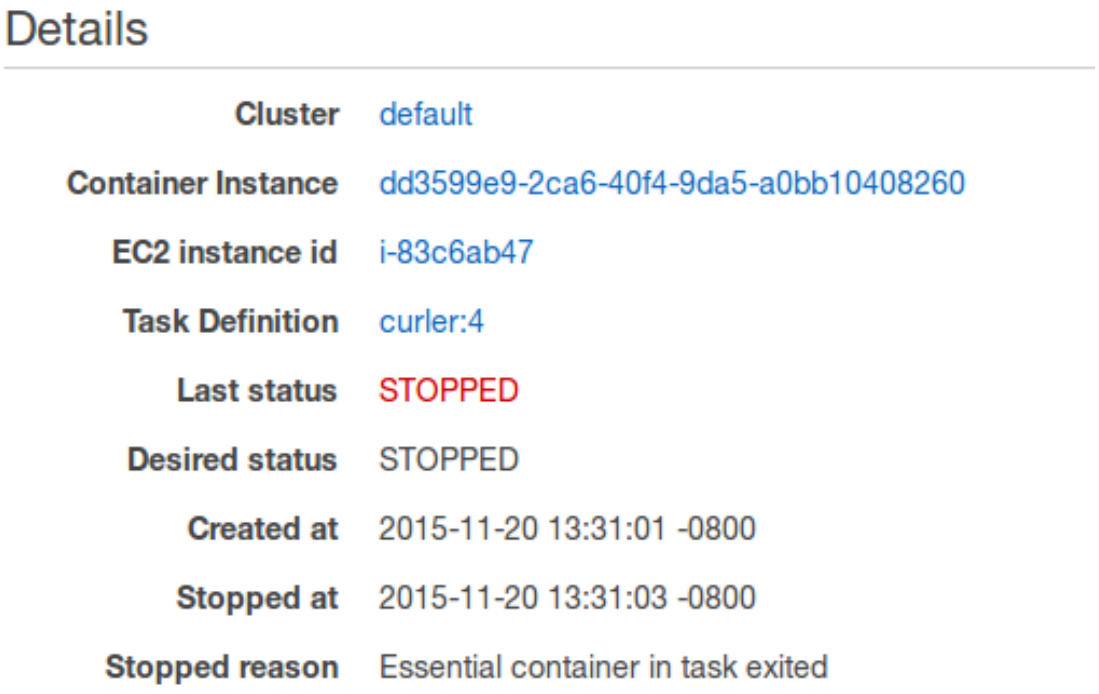
- Checking stopped tasks for errors (p. 690)
- CannotPullContainer task errors (p. 692)
- Service Event Messages (p. 694)
- Invalid CPU or memory value specified (p. 697)
- CannotCreateContainerError: API error (500): devmapper (p. 698)
- Troubleshooting service load balancers (p. 699)
- Enabling Docker debug output (p. 700)
- Amazon ECS Log File Locations (p. 701)
- Amazon ECS logs collector (p. 704)
- Agent introspection diagnostics (p. 705)
- Docker diagnostics (p. 707)
- API error messages (p. 709)
- Troubleshooting IAM Roles for Tasks (p. 710)

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.
3. On the **Cluster : *clustername*** page, choose **Tasks**.
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. 699).

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.

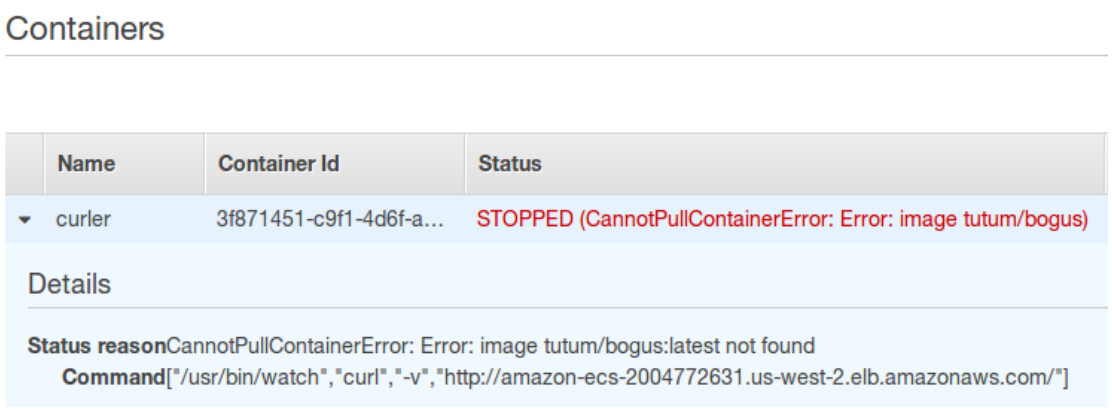
Container instance deregistration forced by user

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. 691) 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.



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. 708).

CannotPullContainer task errors

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:

```
CannotPullContainerError: API error (500): Get https://111122223333.dkr.ecr.us-
east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection"
```

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. 316).
- 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. 643).

Context canceled

The common cause for this error is because the VPC your task is using does not have a route to pull the container image from Amazon ECR.

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. 482).

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/GetImageBlob111111111: 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/
c8e3010e36ce4c089bf286a623699f5233097ca126ebd5a700af023a5127633d/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:

```
CONTAINER ID   IMAGE                               COMMAND             CREATED
 STATUS              PORTS                               NAMES
50501b5f4cbf   amazon/amazon-ecs-agent:latest   "/agent"            4 days ago
 Up 4 days                                           ecs-agent
```

By default, when using the `json-file` log driver, Docker captures the standard output (and standard error) of all of your containers and writes them in files using the JSON format. You 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:

```
{
    "log-driver": "json-file",
    "log-opts": {
        "max-size": "256m"
    }
}
```

An alternative if your container logs are taking up too much disk space is to use the `awslogs` log driver. The `awslogs` log driver sends the logs to CloudWatch, which frees up the disk space that

would otherwise be used for your container logs on the container instance. For more information, see Using the awslogs Log Driver (p. 143).

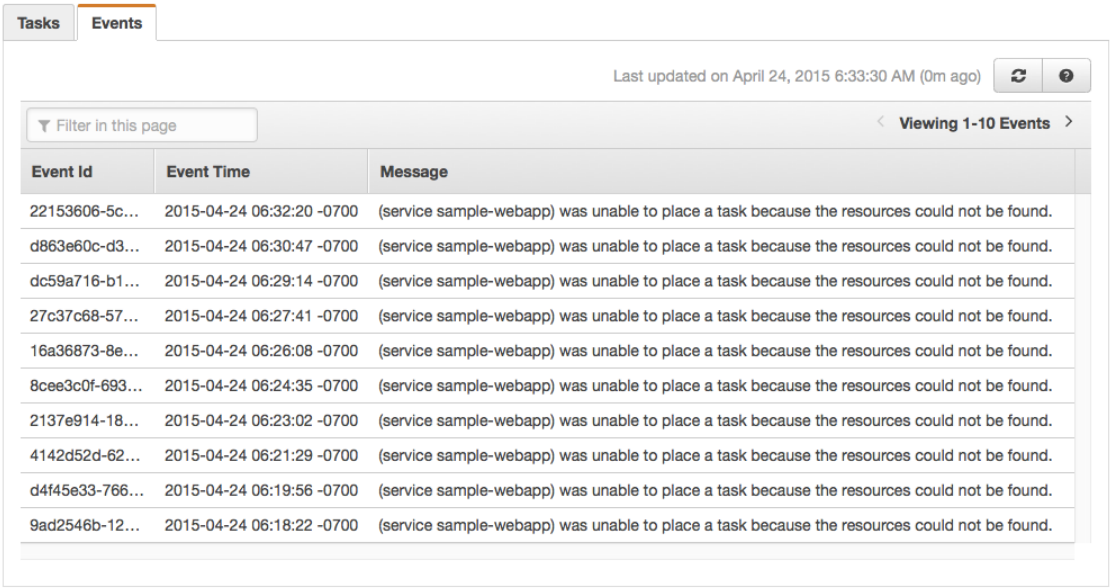
Service Event Messages

If you are troubleshooting a problem with a service, the first place you should check for diagnostic information is the service event log.

When viewing service event messages in the Amazon ECS console, duplicate service event messages are omitted until either the cause is resolved or six hours passes. If the cause is not resolved, you will receive another service event message after six hours has passed.

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.
4. On the **Service : *servicename*** page, choose **Events**.



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 no container instance met all of its requirements. (p. 695)
- service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* has insufficient CPU units available. (p. 696)
- 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. 696)

- 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. 697)
- service (*service-name*) is unable to consistently start tasks successfully. (p. 697)

service (*service-name*) has reached a steady state.

The service scheduler will send a `service` *service-name*`) has reached a steady state.` service event when the service is healthy and at the desired number of tasks, thus reaching a steady state.

service (*service-name*) was unable to place a task because no container instance met all of its requirements.

The service scheduler will send this event message when it could not find the available resources to add another task. The possible causes for this are:

No container instances were found in your cluster

If no container instances are registered in the cluster you attempt to run a task in, you will receive this error. You should add container instances to your cluster. For more information, see Launching an Amazon ECS Container Instance (p. 219).

Not enough ports

If your task uses fixed host port mapping (for example, your task uses port 80 on the host for a web server), you must have at least one container instance per task, because only one container can use a single host port at a time. You should add container instances to your cluster or reduce your number of desired tasks.

Too many ports registered

The closest matching container instance for task placement can not exceed the maximum allowed reserved port limit of 100 host ports per container instance. Using Dynamic host port mapping may remediate the issue.

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. 240).

Not enough CPU

A container instance has 1,024 CPU units for every CPU core. If your task definition specifies 1,000 CPU units, and the container instances in your cluster each have 1,024 CPU units, you can only run one copy of this task per container instance. You can experiment with fewer CPU units in your task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

Not enough available ENI attachment points

Tasks that use the `awsvpc` network mode each receive their own elastic network interface (ENI), which is attached to the container instance that hosts it. Amazon EC2 instances have a limit to the number of ENIs that can be attached to them and there are no container instances in the cluster that have ENI capacity available.

The ENI limit for individual container instances depends on the following conditions:

- If you **have not** opted in to the `awsvpcTrunking` account setting, the ENI limit for each container instance depends on the instance type. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.
- If you **have** opted in to the `awsvpcTrunking` account setting but you **have not** launched new container instances using a supported instance type after opting in, the ENI limit for each container instance will still be at the default value. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.
- If you **have** opted in to the `awsvpcTrunking` account setting and you **have** launched new container instances using a supported instance type after opting in, additional ENIs are available. For more information, see Supported Amazon EC2 Instance Types (p. 234).

For more information about opting in to the `awsvpcTrunking` account setting, see Elastic Network Interface Trunking (p. 230).

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.

A common cause of this error is if your service is using tasks that use the `awsvpc` network mode and the EC2 launch type and the cluster you specified does not have a container instance registered to it in the same subnet that was specified in the `awsvpcConfiguration` when the service was created.

For more information on which attributes are required for specific task definition parameters and agent configuration variables, see Task definition parameters (p. 84) and Amazon ECS Container Agent Configuration (p. 270).

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* has insufficient CPU units available.

The closest matching container instance for task placement does not container enough CPU units to meet the requirements in the task definition. Review the CPU requirements in both the task size and container definition parameters of the task definition.

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* encountered error "AGENT".

The Amazon ECS container agent on the closest matching container instance for task placement is disconnected. If you can connect to the container instance with SSH, you can examine the agent logs; for more information, see Amazon ECS Container Agent Log (p. 701). You should also verify that the agent is running on the instance. If you are using the Amazon ECS-optimized AMI, you can try stopping and restarting the agent with the following command:

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs && sudo start ecs
```

service (*service-name*) (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. 699).

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. 406).

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.

CPU value	Memory value (MiB)
256 (.25 vCPU)	512 (0.5GB), 1024 (1GB), 2048 (2GB)

CPU value	Memory value (MiB)
512 (.5 vCPU)	1024 (1GB), 2048 (2GB), 3072 (3GB), 4096 (4GB)
1024 (1 vCPU)	2048 (2GB), 3072 (3GB), 4096 (4GB), 5120 (5GB), 6144 (6GB), 7168 (7GB), 8192 (8GB)
2048 (2 vCPU)	Between 4096 (4GB) and 16384 (16GB) in increments of 1024 (1GB)
4096 (4 vCPU)	Between 8192 (8GB) and 30720 (30GB) in increments of 1024 (1GB)

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.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. 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. 207).

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. 270).
- 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 Service Scheduler IAM Role (p. 479).

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. 236).
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 service ecs restart
```

Your Docker logs should now show more verbose output. For example:

```
time="2015-12-30T21:48:21.907640838Z" level=debug msg="Unexpected response from
 server: \"{\\\"errors\\\":[{\\\"code\\\":\\\"DENIED\\\",\\\"message\\\":\\\"User:
 arn:aws:sts::1111:assumed-role/ecrReadOnly/i-abcdefg is not authorized to perform:
 ecr:InitiateLayerUpload on resource: arn:aws:ecr:us-east-1:1111:repository/nginx_test
\\\"}]}\\n\" http.Header{\"Connection\":[]string{\"keep-alive\"}, \"Content-Type\":
[]string{\"application/json; charset=utf-8\"}, \"Date\":[]string{\"Wed, 30 Dec 2015
 21:48:21 GMT\"}, \"Docker-Distribution-Api-Version\":[]string{\"registry/2.0\"},
 \"Content-Length\":[]string{\"235\"}}"
```

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 from the `ecs-init` service that controls the state of the agent (start/stop) on the container instance. You can view these log files by connecting to a container instance using SSH. For more information, see Connect to Your Container Instance (p. 236).

Note
If you are not sure how to collect all of the logs on your container instances, you can use the Amazon ECS logs collector. For more information, see Amazon ECS logs collector (p. 704).

Amazon ECS Container Agent Log

The Amazon ECS container agent stores logs on your container instances.

For container agent version 1.36.0 and later, by default the logs are located at `/var/log/ecs/ecs-agent.log` on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log` on Windows instances.

For container agent version 1.35.0 and earlier, by default the logs are located at `/var/log/ecs/ecs-agent.log.`*`timestamp`* on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log.`*`timestamp`* on Windows instances.

By default, the agent logs are rotated hourly with a maximum of 24 logs being stored.

The following are the container agent configuration variables that can be used to change the default agent logging behavior. For more information, see Amazon ECS Container Agent Configuration (p. 270).

`ECS_LOGFILE`

Example values: `/ecs-agent.log`

Default value on Linux: Null

Default value on Windows: Null

The location where agent logs should be written. If you are running the agent via `ecs-init`, which is the default method when using the Amazon ECS-optimized AMI, the in-container path will be `/log` and `ecs-init` mounts that out to `/var/log/ecs/` on the host.

`ECS_LOGLEVEL`

Example values: `crit`, `error`, `warn`, `info`, `debug`

Default value on Linux: `info`

Default value on Windows: `info`

The level of detail to log.

`ECS_LOG_ROLLOVER_TYPE`

Example values: `size`, `hourly`

Default value on Linux: `hourly`

Default value on Windows: `hourly`

Determines whether the container agent log file will be rotated hourly or based on size. By default, the agent log file is rotated each hour.

`ECS_LOG_OUTPUT_FORMAT`

Example values: `logfmt`, `json`

Default value on Linux: `logfmt`

Default value on Windows: `logfmt`

Determines the log output format. When the `json` format is used, each line in the log will be a structured JSON map.

`ECS_LOG_MAX_FILE_SIZE_MB`

Example values: `10`

Default value on Linux: `10`

Default value on Windows: `10`

When the `ECS_LOG_ROLLOVER_TYPE` variable is set to `size`, this variable determines the maximum size (in MB) of the log file before it is rotated. If the rollover type is set to `hourly`, then this variable is ignored.

`ECS_LOG_MAX_ROLL_COUNT`

Example values: `24`

Default value on Linux: `24`

Default value on Windows: `24`

Determines the number of rotated log files to keep. Older log files are deleted once this limit is reached.

For container agent version 1.36.0 and later, the following is an example log file when the `logfmt` format is used.

```
level=info time=2019-12-12T23:43:29Z msg="Loading configuration" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-ecs-
agent:latest" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-ecs-
pause:0.1.0" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Amazon ECS agent Version: 1.36.0, Commit:
 ca640387" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Creating root ecs cgroup: /ecs"
 module=init_linux.go
level=info time=2019-12-12T23:43:29Z msg="Creating cgroup /ecs"
 module=cgroup_controller_linux.go
level=info time=2019-12-12T23:43:29Z msg="Loading state!" module=statemanager.go
level=info time=2019-12-12T23:43:29Z msg="Event stream ContainerChange start listening..."
 module=eventstream.go
level=info time=2019-12-12T23:43:29Z msg="Restored cluster 'auto-robc'" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Restored from checkpoint file. I
 am running as 'arn:aws:ecs:us-west-2:0123456789:container-instance/auto-
robc/3330a8a91d15464ea30662d5840164cd' in cluster 'auto-robc'" module=agent.go
```

The following is an example log file when the JSON format is used.

```
{"time": "2019-11-07T22:52:02Z", "level": "info", "msg": "Starting Amazon Elastic Container
 Service Agent", "module": "engine.go"}
```

For container agent versions 1.35.0 and earlier, the following is the format of the log file.

```
2016-08-15T15:54:41Z [INFO] Starting Agent: Amazon ECS Agent - v1.12.0 (895f3c1)
2016-08-15T15:54:41Z [INFO] Loading configuration
2016-08-15T15:54:41Z [WARN] Invalid value for task cleanup duration, will be overridden to
 3h0m0s, parsed value 0, minimum threshold 1m0s
2016-08-15T15:54:41Z [INFO] Checkpointing is enabled. Attempting to load state
2016-08-15T15:54:41Z [INFO] Loading state! module="statemanager"
2016-08-15T15:54:41Z [INFO] Detected Docker versions [1.17 1.18 1.19 1.20 1.21 1.22]
2016-08-15T15:54:41Z [INFO] Registering Instance with ECS
2016-08-15T15:54:41Z [INFO] Registered! module="api client"
```

Amazon ECS ecs-init Log

The ecs-init process stores logs at /var/log/ecs/ecs-init.log.

```
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 saved in an audit log. The audit log inherits the same log rotation settings as the container agent log. The ECS_LOG_ROLLOVER_TYPE, ECS_LOG_MAX_FILE_SIZE_MB, and ECS_LOG_MAX_ROLL_COUNT container agent configuration variables can be set to affect the behavior of the audit log. For more information, see Amazon ECS Container Agent Log (p. 701).

For container agent version 1.36.0 and later, the audit log is located at `/var/log/ecs/audit.log`. When the log is rotated, a timestamp in *YYYY*-*MM*-*DD*-*HH* format is added to the end of the log file name.

For container agent version 1.35.0 and earlier, the audit log is located at `/var/log/ecs/audit.log.`*YYYY*-*MM*-*DD*-*HH*.

The log entry format is as follows:

- Timestamp
- HTTP response code
- IP address and port number of request origin
- Relative URI of the credential provider
- The user agent that made the request
- The ARN of the task to which the requesting container belongs
- The `GetCredentials` API name and version number
- The name of the Amazon ECS cluster to which the container instance is registered
- The container instance ARN

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. 236).
2. Download the Amazon ECS logs collector script.

```
curl -O https://raw.githubusercontent.com/awslabs/ecs-logs-collector/master/ecs-logs-collector.sh
```

3. Run the script to collect the logs and create the archive.

   Note
   To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `--mode=enable-debug` option to the 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. 239).

```
[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.

```
Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-logs-collector.ps1
```

3. Run the script to collect the logs and create the archive.

   Note
   To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `-RunMode debug` option to the 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. 239).

```
.\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. 236).

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. 486).

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 -mjson.tool** for greater readability.

```
curl http://localhost:51678/v1/tasks | python -mjson.tool
```

Output:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1095  100  1095    0     0   117k      0 --:--:-- --:--:-- --:--:--  133k
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/090eff9b-1ce3-4db6-848a-
a8d14064fd24",
            "Containers": [
                {
                    "DockerId":
 "189a8ff4b5f04affe40e5160a5ffadca395136eb5faf4950c57963c06f82c76d",
                    "DockerName": "ecs-console-sample-app-static-6-simple-
app-86caf9bcabe3e9c61600",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "f7f1f8a7a245c5da83aa92729bd28c6bcb004d1f6a35409e4207e1d34030e966",
                    "DockerName": "ecs-console-sample-app-static-6-busybox-
ce83ce978a87a890ab01",
                    "Name": "busybox"
                }
            ],
            "Family": "console-sample-app-static",
            "KnownStatus": "STOPPED",
            "Version": "6"
        },
        {
            "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/1810e302-eaea-4da9-
a638-097bea534740",
            "Containers": [
                {
                    "DockerId":
 "dc7240fe892ab233dbbcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
                    "DockerName": "ecs-console-sample-app-static-6-simple-app-
f0e5859699a7aecfb101",
                    "Name": "simple-app"
                },
                {
                    "DockerId":
 "096d685fb85a1ff3e021c8254672ab8497e3c13986b9cf005cbae9460b7b901e",
                    "DockerName": "ecs-console-sample-app-static-6-
busybox-92e4b8d0ecd0cce69a01",
                    "Name": "busybox"
                }
            ],
```

```
            "DesiredStatus": "RUNNING",
            "Family": "console-sample-app-static",
            "KnownStatus": "RUNNING",
            "Version": "6"
        }
    ]
}
```

In the above example, the stopped task (*090eff9b-1ce3-4db6-848a-a8d14064fd24*) has two containers. You can use **docker inspect** ***container-ID*** to view detailed information on each container. For more information, see Amazon ECS Container Agent Introspection (p. 309).

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. 236).

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:

```
CONTAINER ID        IMAGE                            COMMAND             CREATED
  STATUS              PORTS                          NAMES
cee0d6986de0        amazon/amazon-ecs-agent:latest   "/agent"            22 hours ago
  Up 22 hours         127.0.0.1:51678->51678/tcp     ecs-agent
```

You can use the **docker ps -a** command to see all containers (even stopped or killed containers). This is helpful for listing containers that are unexpectedly stopping. In the following example, container `f7f1f8a7a245` exited 9 seconds ago, so it does not show up in a **docker ps** output without the `-a` flag.

```
docker ps -a
```

Output:

```
CONTAINER ID        IMAGE                                          COMMAND
 CREATED             STATUS                          PORTS                          NAMES
db4d48e411b1        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"       19
 seconds ago                                                                        ecs-console-
sample-app-static-6-internalecs-emptyvolume-source-c09288a6b0cba8a53700
f7f1f8a7a245        busybox:buildroot-2014.02                   "\"sh -c '/bin/sh -c   22
 hours ago        Exited (137) 9 seconds ago                                        ecs-console-
sample-app-static-6-busybox-ce83ce978a87a890ab01
```

```
189a8ff4b5f0        httpd:2                                          "httpd-foreground"     22
 hours ago        Exited (137) 40 seconds ago                                 ecs-console-
sample-app-static-6-simple-app-86caf9bcabe3e9c61600
0c7dca9321e3        amazon/ecs-emptyvolume-base:autogenerated   "not-applicable"       22
 hours ago                                                                   ecs-console-
sample-app-static-6-internalecs-emptyvolume-source-90fefaa68498a8a80700
cee0d6986de0        amazon/amazon-ecs-agent:latest              "/agent"               22
 hours ago        Up 22 hours                   127.0.0.1:51678->51678/tcp   ecs-agent
```

View Docker Logs

You can view the `STDOUT` and `STDERR` streams for a container with the **docker logs** command. In this example, the logs are displayed for the *dc7240fe892a* container and piped through the **head** command for brevity. For more information, go to docker logs in the Docker documentation.

> **Note**
> Docker logs are only available on the container instance if you are using the default `json` log driver. If you have configured your tasks to use the `awslogs` log driver, then your container logs are available in CloudWatch Logs. For more information, see Using the awslogs Log Driver (p. 143).

```
docker logs dc7240fe892a | head
```

Output:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
[Thu Apr 23 19:48:36.956682 2015] [mpm_event:notice] [pid 1:tid 140327115417472] AH00489:
 Apache/2.4.12 (Unix) configured -- resuming normal operations
[Thu Apr 23 19:48:36.956827 2015] [core:notice] [pid 1:tid 140327115417472] AH00094:
 Command line: 'httpd -D FOREGROUND'
10.0.1.86 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:29 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.0.154 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.1.86 - - [23/Apr/2015:19:49:58 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:50:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:50:29 +0000] "GET / HTTP/1.1" 200 348
time="2015-04-23T20:11:20Z" level="fatal" msg="write /dev/stdout: broken pipe"
```

Inspect Docker Containers

If you have the Docker ID of a container, you can inspect it with the **docker inspect** command. Inspecting containers provides the most detailed view of the environment in which a container was launched. For more information, see docker inspect in the Docker documentation.

```
docker inspect dc7240fe892a
```

Output:

```
[{
```

```
    "AppArmorProfile": "",
    "Args": [],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "httpd-foreground"
        ],
        "CpuShares": 10,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/
apache2/bin",
            "HTTPD_PREFIX=/usr/local/apache2",
            "HTTPD_VERSION=2.4.12",
            "HTTPD_BZ2_URL=https://www.apache.org/dist/httpd/httpd-2.4.12.tar.bz2"
        ],
        "ExposedPorts": {
            "80/tcp": {}
        },
        "Hostname": "dc7240fe892a",
...
```

API 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. 121).

  `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.

  `LOCATION` (container instance ID)

  The container instance that you attempted to launch a task onto is in a different Availability Zone than the subnet(s) you specified in your `awsVpcConfiguration`.

  `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. 84) and Amazon ECS Container Agent Configuration (p. 270).

- `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`.
   a. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
   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.
   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   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.

```
{
    "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"
                }
            }
        }
    ],
```

```
    "family": "iam-role-test",
    "requiresCompatibilities": [
        "EC2"
    ],
    "volumes": [],
    "placementConstraints": [],
    "networkMode": null,
    "memory": null,
    "cpu": null
}
```

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

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.

   a. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

   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.

```
{
    "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. 491).
   - 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. 270).

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. For more information, see Amazon ECS-optimized AMIs (p. 190).

- **Amazon ECS-optimized Windows 2019 Full AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows 2019 Core AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows 1909 Core AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows 2016 Full AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.

Topics

- Windows container caveats (p. 713)
- Getting started with Windows containers (p. 714)
- Windows task definitions (p. 718)
- Windows IAM roles for tasks (p. 721)
- Pushing Windows images to Amazon ECR (p. 722)
- Using gMSAs for Windows Containers (p. 723)

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. 119).
- 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. 718).

- 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. 721).
- 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. 376).
- 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 getting Windows containers running on Amazon ECS with the Amazon ECS-optimized Windows AMI in the AWS Management Console. 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. 714)
- Step 2: Register a Windows task definition (p. 716)
- Step 3: Create a service with your task definition (p. 717)
- Step 4: View your service (p. 717)

Step 1: Create a Windows cluster

You can create a new cluster for your Windows containers. Amazon EC2 instances using the Linux Amazon ECS-optimized AMIs 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` and register one or more Amazon EC2 instances into the cluster 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**.
4. Choose **EC2 Windows + Networking** and choose **Next step**.
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. In the **Instance configuation** section, complete the following steps.

    a. 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.

b. For Spot Instances, do the following; otherwise, skip to the next step.

   i. For **Spot Instance allocation strategy**, choose the strategy that meets your needs. For more information, see Spot Fleet Allocation Strategy.

   ii. 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.

c. For **EC2 instance type** page, choose the hardware configuration of your instance. The instance type that you select determines the resources available for your tasks to run on.

d. For **Number of instances**, choose the number of Amazon EC2 instances to launch into your cluster.

e. For **EC2 AMI Id**, choose the Amazon ECS-optimized AMI to use for your container instances. The available AMIs will be determined by the Region and instance type you chose. For more information, see Amazon ECS-optimized AMIs (p. 190).

f. 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.

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

7. 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 Windows instances in the *Amazon EC2 User Guide for Windows 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
      The IAM role you use must have the `AmazonEC2ContainerServiceforEC2Role` managed policy attached to it, otherwise you will receive an error during cluster creation. 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. 486).

f. If you chose the Spot Instance type earlier, the **Spot Fleet Role IAM role** section indicates that an IAM role `ecsSpotFleetRole` is created.

8. 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.
9. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).
10. Choose **Create**.

Note
It can take up to 15 minutes for your Windows container instances to register with your cluster.

Step 2: Register a Windows task definition

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

```
{
  "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 ; C:\
\ServiceMonitor.exe w3svc"],
      "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.

```
aws ecs register-task-definition --cli-input-json file://windows-simple-iis.json
```

Step 3: Create a service with your task definition

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

To create a service from your task definition with the console

1. On the **Task Definition: windows-simple-iis** registration confirmation page, choose **Actions**, **Create Service**.
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.

```
aws ecs create-service --cluster windows --task-definition windows-simple-iis --
desired-count 1 --service-name windows-simple-iis
```

Step 4: View your service

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

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

To view your service

1. Open the 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. 84).

`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. 721).

`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.

`linuxParameters`

Supported: No

`privileged`

Supported: No

`readonlyRootFilesystem`

Supported: No

`user`

Supported: No

`ulimits`

Supported: No

`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

Windows sample task definitions

Below is a sample task definition to help you get started with Windows containers on Amazon ECS.

Example Amazon ECS Console Sample Application for Windows

The following task definition is the Amazon ECS console sample application that is produced in the first-run wizard for Amazon ECS; it has been ported to use the `microsoft/iis` Windows container image.

```
{
  "family": "windows-simple-iis",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "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. 721).
- 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. 491).
- Your container must use an AWS SDK that supports IAM roles for tasks. For more information, see Using a Supported AWS SDK (p. 493).
- 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. 493).
- 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. 376).
- If your Windows instance is restarted, you must delete the proxy interface and initialize the Amazon ECS container agent again to bring the credential proxy back up.

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.

```
# 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.

```
PS C:\> docker pull microsoft/iis
Using default tag: latest
latest: Pulling from microsoft/iis

3889bb8d808b: Pull complete
04ee5d718c7a: Pull complete
c0931dd15237: Pull complete
61784b745c20: Pull complete
d05122f129ca: Pull complete
Digest: sha256:25586570b058da9882d4af640d326d0cc26dfd60b67e1cee63f35ea54d83c882
Status: Downloaded newer image for microsoft/iis:latest
```

2. Create an Amazon ECR repository for your image.

```
PS C:\> aws ecr create-repository --repository-name iis
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "iis",
        "repositoryArn": "arn:aws:ecr:region:aws_account_id:repository/iis",
        "createdAt": 1481845593.0,
        "repositoryUri": "aws_account_id.dkr.ecr.region.amazonaws.com/iis"
    }
}
```

3. Tag the image with the `repositoryUri` that was returned from the previous command.

```
PS C:\> docker tag microsoft/iis aws_account_id.dkr.ecr.region.amazonaws.com/iis
```

4. To authenticate Docker to an Amazon ECR registry with get-login-password, run the **aws ecr get-login-password** command. When passing the authentication token to the **docker login** command, you specify the `AWS` username and your Amazon ECR registry URI.

Important
If you receive an error, install or upgrade to 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:\> aws ecr get-login-password --region region | docker login --username AWS --
password-stdin aws_account_id.dkr.ecr.region.amazonaws.com/iis
```

5. Push the image to Amazon ECR.

```
PS C:\> docker push aws_account_id.dkr.ecr.region.amazonaws.com/iis
The push refers to a repository [111122223333.dkr.ecr.us-west-2.amazonaws.com/iis]
1e4f77a75bd4: Pushed
ac90fb7da567: Pushed
c7090349c7b3: Pushed
b9454c3094c6: Skipped foreign layer
3fd27ecef6a3: Skipped foreign layer
latest: digest: sha256:0ddc7af8691072bb2dd8b3f189388b33604c90774d3dc0485b1bf379f9bec4c5
 size: 1574
```

Using gMSAs for Windows Containers

Amazon ECS supports Active Directory authentication for Windows containers through a special kind of service account called a *group Managed Service Account* (gMSA).

Windows based network applications such as .NET applications often use Active Directory to facilitate authentication and authorization management between users and services. Developers commonly design their applications to integrate with Active Directory and run on domain-joined servers for this purpose. Because Windows containers cannot be domain-joined, you must configure a Windows container to run with gMSA.

A Windows container running with gMSA relies on its host Amazon EC2 instance to retrieve the gMSA credentials from the Active Directory domain controller and provide them to the container instance. For more information, see Create gMSAs for Windows containers.

Topics

- Considerations (p. 723)
- Prerequisites (p. 723)
- Setting Up gMSA-capable Windows Containers on Amazon ECS (p. 724)

Considerations

The following should be considered when using gMSAs for Windows containers:

- When using the Amazon ECS-optimized Windows 2016 Full AMI for your container instances, the container hostname must be the same as the gMSA account name defined in the credential spec file. To specify a hostname for a container, use the `hostname` container definition parameter. For more information, see Network Settings (p. 95).

Prerequisites

The following are prerequisites for using the gMSA for Windows containers feature with Amazon ECS.

- An Active Directory that your Amazon ECS Windows container instances can join. Amazon ECS supports the following:
  - AWS Directory Service, which is an AWS managed Active Directory hosted on Amazon EC2. For more information, see Getting Started with AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.
  - On-premises Active Directory, as long as the Amazon ECS Windows container instance can join the domain. For more information, see AWS Direct Connect.
- An existing gMSA account in the Active Directory. For more information, see Create gMSAs for Windows containers.
- The Amazon ECS Windows container instance hosting the Amazon ECS task must be domain joined to the Active Directory and be a member of the Active Directory security group that has access to the gMSA account.

Setting Up gMSA-capable Windows Containers on Amazon ECS

Amazon ECS uses a credential spec file that contains the gMSA metadata used to propagate the gMSA account context to the Windows container. You can generate the credential spec file and reference it in the `dockerSecurityOptions` field in your task definition. The credential spec file does not contain any secrets.

The following is an example credential spec file:

```
{
    "CmsPlugins": [
        "ActiveDirectory"
    ],
    "DomainJoinConfig": {
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",
        "MachineAccountName": "WebApp01",
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",
        "DnsTreeName": "example.com",
        "DnsName": "example.com",
        "NetBiosName": "example"
    },
    "ActiveDirectoryConfig": {
        "GroupManagedServiceAccounts": [
            {
                "Name": "WebApp01",
                "Scope": "example.com"
            }
        ]
    }
}
```

Referencing a Credential Spec File in a Task Definition

Amazon ECS supports the following ways to reference the credential spec file in the `dockerSecurityOptions` field of a task definition.

Topics

- Amazon S3 Bucket (p. 725)
- SSM Parameter Store parameter (p. 725)
- Local File (p. 726)

Amazon S3 Bucket

Add the credential spec to an Amazon S3 bucket and then reference the Amazon Resource Name (ARN) of the Amazon S3 bucket in the `dockerSecurityOptions` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "dockerSecurityOptions": [
                "credentialspec:arn:aws:s3:::${BucketName}/${ObjectName}"
            ],
            ...
        }
    ],
    ...
}
```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the Amazon S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor",
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": [
                "arn:aws:s3:::{bucket_name}",
                "arn:aws:s3:::{bucket_name}/{object}"
            ]
        }
    ]
}
```

SSM Parameter Store parameter

Add the credential spec to an SSM Parameter Store parameter and then reference the Amazon Resource Name (ARN) of the SSM Parameter Store parameter in the `dockerSecurityOptions` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "dockerSecurityOptions": [
                "credentialspec:arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ],
            ...
        }
    ],
```

```
    ...
}
```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the SSM Parameter Store parameter.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ]
        }
    ]
}
```

Local File

With the credential spec details in a local file, reference the file path in the `dockerSecurityOptions` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "dockerSecurityOptions": [
                "credentialspec:file://CredentialSpecFile.json"
            ],
            ...
        }
    ],
    ...
}
```

Document History

The following table describes the major updates and new features for the *Amazon Elastic Container Service Developer Guide*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
AWS Fargate platform version 1.4.0 update	Beginning on May 28, 2020, any new Fargate task that is launched using platform version 1.4.0 will have its 20 GB ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key. For more information, see Fargate Task Storage (p. 125).	28 May 2020
Environment variable file support	Added support for specifying environment variable files in a task definition, which enables you to bulk add environment variables to your containers. For more information, see Specifying environment variables (p. 175).	18 May 2020
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Africa (Cape Town) Region.	11 May 2020
Service quota updated	The following service quota was updated: • Clusters per account was raised from `2,000` to `10,000`. For more information, see Amazon ECS service quotas (p. 612).	17 April 2020
AWS Fargate platform version 1.4.0	AWS Fargate platform version 1.4.0 is released, which contains the following features: • Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see Amazon EFS Volumes (p. 138). • The ephemeral task storage has been increased to 20 GB. For more information, see Fargate Task Storage (p. 125). • The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see Fargate Task Networking in the *Amazon Elastic Container Service User Guide for AWS Fargate*. • Task ENIs add support for jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single	8 April 2020

Change	Description	Date
	frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames will reduce overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC. • CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see Amazon ECS CloudWatch Container Insights (p. 439). • Added support for the task metadata endpoint v4 which provides additional information for your Fargate tasks, including network stats for the task and which Availability Zone the task is running in. For more information, see Task Metadata Endpoint version 4 (p. 292). • Added support for the `SYS_PTRACE` Linux parameter in container definitions. For more information, see Linux Parameters (p. 103). • The Fargate container agent replaces the use of the Amazon ECS container agent for all Fargate tasks. This change should not have an effect on how your tasks run. • The container runtime is now using Containerd instead of Docker. This change should not have an effect on how your tasks run. You will notice that some error messages that originate with the container runtime will change from mentioning Docker to more general errors. For more information, see AWS Fargate Platform Versions (p. 33).	
Amazon EFS file system support for task volumes	Amazon EFS file systems can be used as data volumes for both your Amazon ECS and Fargate tasks. For more information, see Amazon EFS Volumes (p. 138).	8 April 2020
Amazon ECS Task Metadata Endpoint version 4	Beginning with Amazon ECS container agent version 1.39.0 and Fargate platform version 1.4.0, an environment variable named `ECS_CONTAINER_METADATA_URI_V4` is injected into each container in a task. When you query the task metadata version 4 endpoint, various task metadata and Docker stats are available to tasks. For more information, see Task Metadata Endpoint version 4 (p. 292).	8 April 2020
Support for specific versions of Secrets Manager secrets to be injected as environment variables	Added support for specifying sensitive data using specific versions of Secrets Manager secrets. For more information, see Injecting Sensitive Data as an Environment Variable (p. 165).	24 Feb 2020

Change	Description	Date
Added additional CodeDeploy deployment configuration options for blue/green deployments	The CodeDeploy service added new canary and linear deployment configurations for the Amazon ECS deployment type. The ability to define custom deployment configurations is also available. For more information, see Blue/Green Deployment with CodeDeploy (p. 367).	6 Feb 2020
Added the efsVolumeConfiguration task definition parameter	The `efsVolumeConfiguration` task definition parameter is in public preview, which makes it easier to use Amazon EFS file systems with your Amazon ECS tasks. For more information, see Amazon EFS Volumes (p. 138).	17 Jan 2020
Amazon ECS container agent logging behavior updated	The Amazon ECS container agent logging locations and rotation behavior has been updated. For more information, see Amazon ECS Container Agent Log (p. 701).	13 Jan 2020
Fargate Spot	Amazon ECS added support for running tasks using Fargate Spot. For more information, see Using AWS Fargate Capacity Providers (p. 67).	3 Dec 2019
Cluster Auto Scaling	Amazon ECS cluster auto scaling enables you to have more control over how you scale tasks within a cluster. For more information, see Amazon ECS Cluster Auto Scaling (p. 42).	3 Dec 2019
Cluster Capacity Providers	Amazon ECS cluster capacity providers determine the infrastructure to use for your tasks. For more information, see Amazon ECS Cluster Capacity Providers (p. 40).	3 Dec 2019
Creating a cluster on an AWS Outposts	Amazon ECS now supports creating clusters on an AWS Outposts. For more information, see *Amazon Elastic Container Service on AWS Outposts* (p. 619).	3 Dec 2019
Service Action Events	Amazon ECS now sends events to Amazon EventBridge when certain service actions occur. For more information, see Service Action Events (p. 434).	25 Nov 2019
Savings Plans	Savings Plans are a pricing model that offer significant savings on AWS usage. For more information, see Savings Plans and Amazon ECS (p. 618).	6 Nov 2019
Amazon ECS GPU-optimized AMI Supports G4 Instances	Amazon ECS added support for the g4 instance type family when using the Amazon ECS GPU-optimized AMI. For more information, see Working with GPUs on Amazon ECS (p. 121).	8 Oct 2019
Amazon ECS CLI v1.17.0	New version of the Amazon ECS CLI released. This release added support for specifying a FireLens configuration using the ECS Parameters file. For more information, see Using Amazon ECS Parameters (p. 606).	2 Oct 2019
FireLens for Amazon ECS	FireLens for Amazon ECS is in general availability. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Custom Log Routing (p. 150).	30 Sept 2019

Change	Description	Date
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Europe (Paris), Europe (Stockholm), and Middle East (Bahrain) regions.	30 Sept 2019
Deep Learning Containers with Elastic Inference on Amazon ECS	Amazon ECS supports attaching Amazon Elastic Inference accelerators to your containers to make running deep learning inference workloads more efficient. For more information, see Deep Learning Containers with Elastic Inference on Amazon ECS (p. 642).	3 Sept 2019
FireLens for Amazon ECS	FireLens for Amazon ECS is in public preview. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Custom Log Routing (p. 150).	30 Aug 2019
CloudWatch Container Insights	CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).	30 Aug 2019
Container Level Swap Configuration	Amazon ECS added support for controlling the usage of swap memory space on your Linux container instances at the container level. Using a per-container swap configuration, each container within a task definition can have swap enabled or disabled, and for those that have it enabled, the maximum amount of swap space used can be limited. For more information, see Managing Container Swap Space (p. 242).	16 Aug 2019
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Hong Kong) Region.	06 Aug 2019
Elastic Network Interface Trunking	Added additional supported Amazon EC2 instance types for ENI trunking feature. For more information, see Supported Amazon EC2 Instance Types (p. 234).	1 Aug 2019
Registering Multiple Target Groups with a Service	Added support for specifying multiple target groups in a service definition. For more information, see Registering Multiple Target Groups with a Service (p. 393).	30 July 2019
Specifying Sensitive Data Using Secrets Manager Secrets	Added tutorial for specifying sensitive data using Secrets Manager secrets. For more information, see Tutorial: Specifying Sensitive Data Using Secrets Manager Secrets (p. 657).	20 July 2019
Amazon ECS CLI v1.15.0	New version of the Amazon ECS CLI released. For more information, see Amazon ECS CLI Changelog.	9 July 2019
CloudWatch Container Insights	Amazon ECS has added support for CloudWatch Container Insights. For more information, see Amazon ECS CloudWatch Container Insights (p. 439).	9 July 2019

Change	Description	Date
Resource-level permissions for Amazon ECS services and tasksets	Amazon ECS has expanded resource-level permissions support for Amazon ECS services and tasks. For more information, see How Amazon Elastic Container Service Works with IAM (p. 448).	27 June 2019
New Amazon ECS-optimized AMI patched for AWS-2019-005	Amazon ECS has updated the Amazon ECS-optimized AMI to address the vulnerabilities described in AWS-2019-005.	17 June 2019
Elastic Network Interface Trunking	Amazon ECS introduces support for launching container instances using supported Amazon EC2 instance types that have increased elastic network interface (ENI) density. Using these instance types and opting in to the `awsvpcTrunking` account setting provides increased ENI density on newly launched container instances which allows you to place more tasks on each container instance. For more information, see Elastic Network Interface Trunking (p. 230).	6 June 2019
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports the `splunk` log driver in addition to the `awslogs` log driver. For more information, see Storage and Logging (p. 97).	1 May 2019
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports referencing sensitive data in the log configuration of a container using the `secretOptions` container definition parameter. For more information, see Specifying Sensitive Data (p. 163).	1 May 2019
AWS Fargate platform version 1.3.0 update	Beginning on April 2, 2019, any new Fargate task that is launched supports injecting sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. For more information, see Specifying Sensitive Data (p. 163).	2 Apr 2019
AWS Fargate platform version 1.3.0 update	Beginning on March 27, 2019, any new Fargate task launched can use additional task definition parameters that enable you to define a proxy configuration, dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Proxy configuration (p. 116), Container Dependency (p. 107), and Container Timeouts (p. 108).	27 Mar 2019
Amazon ECS introduces the external deployment type	The *external* deployment type enables you to use any third-party deployment controller for full control over the deployment process for an Amazon ECS service. For more information, see External Deployment (p. 371).	27 Mar 2019
Getting Started with AWS App Mesh and Amazon ECS	Added tutorial for AWS App Mesh and Amazon ECS. For more information, see Getting started with AWS App Mesh and Amazon ECS (p. 623).	27 Mar 2019

Change	Description	Date
AWS Deep Learning Containers on Amazon ECS	AWS Deep Learning Containers are a set of Docker images for training and serving models in TensorFlow on Amazon Elastic Container Service (Amazon ECS). Deep Learning Containers provide optimized environments with TensorFlow, Nvidia CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries and are available in Amazon ECR. For more information, see AWS Deep Learning Containers on Amazon ECS (p. 642).	27 Mar 2019
Amazon ECS introduces enhanced container dependency management	Amazon ECS introduces additional task definition parameters that enable you to define dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Container Dependency (p. 107).	7 Mar 2019
Amazon ECS CLI v1.13.0	New version of the Amazon ECS CLI released. For more information, see Amazon ECS CLI Changelog.	7 Mar 2019
Amazon ECS introduces the `PutAccountSettingDefault` API	Amazon ECS introduces the `PutAccountSettingDefault` API that allows a user to set the default ARN/ID format opt in status for all the 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. 184).	8 Feb 2019
Amazon ECS supports GPU workloads	Amazon ECS introduces support for GPU workloads by enabling you to create clusters with GPU-enabled container instances. In a task definition you can specify the number of required GPUs and the ECS agent will pin the physical GPUs to the container. For more information, see Working with GPUs on Amazon ECS (p. 121).	4 Feb 2019
Amazon ECS expanded secrets support	Amazon ECS expanded support for using AWS Secrets Manager secrets directly in your task definitions to inject sensitive data into your containers. For more information, see Specifying Sensitive Data (p. 163).	21 Jan 2019
Interface VPC Endpoints (AWS PrivateLink)	Added support for configuring interface VPC endpoints powered by AWS PrivateLink. This allows you to create a private connection between your VPC and Amazon ECS without requiring access over the Internet, through a NAT instance, a VPN connection, or AWS Direct Connect. For more information, see Interface VPC Endpoints (AWS PrivateLink).	26 Dec 2018

Change	Description	Date
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. 163). • 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. 336). For more information, see AWS Fargate Platform Versions (p. 33).	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 quotas (p. 612).	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. 33).	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. 367).	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. 190).	26 Nov 2018

Change	Description	Date
Amazon ECS CLI v1.11.2	New version of the Amazon ECS CLI released, which added the following functionality: • 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. 606). • Added support for specifying the `ipcMode` and `pidMode` Docker flags in task definitions. For more information, see Using Amazon ECS Parameters (p. 606).	19 Nov 2018
Added support for additional Docker flags in task definitions	Amazon ECS introduced support for the following Docker flags in task definitions: • IPC mode (p. 118) • PID mode (p. 118)	16 Nov 2018
Amazon ECS secrets support	Amazon ECS added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers. For more information, see Specifying Sensitive Data (p. 163).	15 Nov 2018
Resource tagging	Amazon ECS added support for adding metadata tags to your services, task definitions, tasks, clusters, and container instances. For more information, see Resources and Tags (p. 407).	15 Nov 2018
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the US West (N. California) and Asia Pacific (Seoul) Regions. For more information, see Amazon ECS on AWS Fargate (p. 27).	07 Nov 2018
Service limits updated	The following service limits were updated: • Number of tasks using the Fargate launch type, per Region, per account was raised from `20` to `50`. • Number of public IP addresses for tasks using the Fargate launch type was raised from `20` to `50`. For more information, see Amazon ECS service quotas (p. 612).	31 Oct 2018
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Europe (London) Region. For more information, see Amazon ECS on AWS Fargate (p. 27).	26 Oct 2018

Change	Description	Date
Amazon ECS CLI v1.10.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added the **ecs-cli registry-creds** command, which facilitates the creation and use of private registry credentials within Amazon ECS. For more information, see ecs-cli registry-creds (p. 592). • Added support for Amazon Linux 2. For more information, see Amazon ECS-optimized AMIs (p. 190).	25 October 2018
Amazon ECS-optimized Amazon Linux 2 AMI Released	Amazon ECS vends Linux AMIs that are optimized for the service in two variants. The latest and recommended version is based on x;. Amazon ECS also vends AMIs that are based on the 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 Amazon ECS-optimized AMIs (p. 190).	18 October 2018
Amazon ECS CLI v1.9.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for service discovery. For more information, see Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 522). • Added support for Amazon EC2 Spot instances in Amazon ECS clusters. • Added support for custom user data.	18 October 2018
Amazon ECS Task Metadata Endpoint version 3	Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called `ECS_CONTAINER_METADATA_URI` into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and Docker stats are available to tasks that use the `awsvpc` network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see Amazon ECS Task Metadata Endpoint (p. 292).	18 October 2018
Amazon ECS service discovery Region expansion	Amazon ECS service discovery has expanded support to the Canada (Central), South America (São Paulo), Asia Pacific (Seoul), Asia Pacific (Mumbai), and Europe (Paris) Regions. For more information, see Service Discovery (p. 403).	27 September 2018
Added support for additional Docker flags in container definitions	Amazon ECS introduced support for the following Docker flags in container definitions: • System Controls (p. 109) • Interactive (p. 110) • Pseudo Terminal (p. 110)	17 Sept 2018

Change	Description	Date
Private registry authentication support for Amazon ECS using AWS Fargate tasks	Amazon ECS introduced support for Fargate tasks using private registry authentication using AWS Secrets Manager. This feature enables you to store your credentials securely and then reference them in your container definition, which allows your tasks to use private images. For more information, see Private Registry Authentication for Tasks (p. 160).	10 Sept 2018
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. 554). • Added support for task placement constraints and strategies in Docker compose files. For more information, see ecs-cli compose (p. 554). • Added support for private registry authentication in Docker compose files. For more information, see ecs-cli compose (p. 554). • Added support for `--force-update` on `compose up` to force relaunching of tasks. For more information, see ecs-cli compose up (p. 563).	7 Sept 2018
Amazon ECS service discovery Region expansion	Amazon ECS service discovery has expanded support to the Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), EU (Frankfurt), and Europe (London) Regions. For more information, see Service Discovery (p. 403).	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. 330).	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. 160).	16 August 2018
Docker volume support added	Amazon ECS introduced support for Docker volumes. For more information, see Using Data Volumes in Tasks (p. 124).	9 August 2018

Change	Description	Date
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Europe (Frankfurt), Asia Pacific (Singapore), and Asia Pacific (Sydney) Regions. For more information, see Amazon ECS on AWS Fargate (p. 27).	19 July 2018
Amazon ECS CLI v1.7.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for container `healthcheck` and `devices` in Docker compose files. For more information, see ecs-cli compose (p. 554).	18 July 2018
Amazon ECS service scheduler strategies added	Amazon ECS introduced the concept of service scheduler strategies. There are two service scheduler strategies available: • `REPLICA`—The replica scheduling strategy places and maintains the desired number of tasks across your cluster. By default, the service scheduler spreads tasks across Availability Zones. You can use task placement strategies and constraints to customize task placement decisions. For more information, see Replica (p. 339). • `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. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. 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. 339). **Note** Fargate tasks do not support the `DAEMON` scheduling strategy. For more information, see Service scheduler concepts (p. 338).	12 June 2018
Amazon ECS CLI v1.6.0	New version of the Amazon ECS CLI released, which added the following functionality: • Added support for Docker compose file syntax version 3. For more information, see ecs-cli compose (p. 554).	5 June 2018

Change	Description	Date
Amazon ECS container agent v1.18.0	New version of the Amazon ECS container agent released, which added the following functionality: • Added 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. 250). • 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. 714). • 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. 270). 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. 403).	22 May 2018
Added support for additional Amazon ECS-optimized AMI metadata parameters	Added subparameters that allow you to programatically retrieve the Amazon ECS-optimized AMI ID, image name, operating system, container agent version, and runtime version. Query the metadata using the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 211).	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 Amazon ECS on AWS Fargate (p. 27).	26 April 2018

Change	Description	Date
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 Systems Manager Parameter Store API during the cluster resource creation process. This requires the user account that you are using to have the required Systems Manager permissions. For more information, see ecs-cli up (p. 533). • Added support for the `shm_size` and `tmpfs` parameters in compose files. For more information, see ecs-cli compose (p. 554). For more information about the updated ECS CLI syntax, see Amazon ECS Command Line Reference (p. 525).	19 April 2018
Amazon ECS-optimized AMI Metadata Retrieval	Added ability to programatically retrieve Amazon ECS-optimized AMI metadata using the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI Metadata (p. 211).	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. 506).	5 April 2018
AWS Fargate Platform Version	New AWS Fargate platform version released, which contains: • Added support for Amazon ECS Task Metadata Endpoint (p. 292). • Added support for Health Check (p. 90). • Added support for Service Discovery (p. 403) For more information, see AWS Fargate Platform Versions (p. 33).	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. 403).	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. 525).	20 March 2018

Change	Description	Date
Docker shm-size and tmpfs support	Added support for the Docker shm-size and tmpfs parameters in Amazon ECS task definitions. For more information about the updated ECS CLI syntax, see Linux Parameters (p. 103).	20 March 2018
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. 566). • Added support for `aws_session_token` in ECS profiles. For more information, see ecs-cli configure profile (p. 530). • 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. 525).	09 March 2018
Container Health Checks	Added support for Docker health checks in container definitions. For more information, see Health Check (p. 90).	08 March 2018
AWS Fargate	Added overview for Amazon ECS with AWS Fargate. For more information, see Amazon ECS on AWS Fargate (p. 27).	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 `awsvpc` 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. 292).	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. 396). 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

Change	Description	Date
Amazon ECS CLI v1.3.0	New version of the Amazon ECS CLI released, which added the following functionality: • Ability to create empty clusters with the up command. • Added `--health-check-grace-period` flag for the compose service up command. • Updated the AMI to `amzn-ami-2017.09.g-amazon-ecs-optimized`. For more information about the updated ECS CLI syntax, see Amazon ECS Command Line Reference (p. 525).	19 January 2018
Docker 17.09 support	Added support for Docker 17.09. For more information, see Amazon ECS-optimized AMIs (p. 190).	18 January 2018
Elastic Load Balancing health check initialization wait period	Added ability to specify a wait period for health checks.	27 December 2017
New service scheduler behavior	Updated information about the behavior for service tasks that fail to launch. Documented new service event message that triggers when a service task has consecutive failures. For more information about this updated behavior, see Additional service concepts (p. 340).	11 January 2018
Task-level CPU and memory	Added support for specifying CPU and memory at the task-level in task definitions. For more information, see TaskDefinition.	12 December 2017
Task execution role	The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so it requires an IAM policy and role for the service to know that the agent belongs to you. The following actions are covered by the task execution role: • Calls to Amazon ECR to pull the container image • Calls to CloudWatch to store container application logs For more information, see Amazon ECS Task Execution IAM Role (p. 482).	7 December 2017
Windows containers support GA	Added support for Windows 2016 containers. For more information, see Windows containers (p. 713).	5 December 2017
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.	29 November 2017

Change	Description	Date
AWS Fargate GA	Added support for launching Amazon ECS services using the Fargate launch type. For more information, see Amazon ECS Launch Types (p. 119).	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 (p. 141).	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.	7 November 2017
Amazon ECS container metadata	Amazon ECS containers are now able to access metadata such as their Docker container or image ID, networking configuration, or Amazon ARNs. For more information, see Amazon ECS Container Metadata File (p. 288).	2 November 2017
Docker 17.06 support	Added support for Docker 17.06. For more information, see Amazon ECS-optimized AMIs (p. 190).	2 November 2017
Support for Docker flags: device and init	Added support for Docker's device and init features in task definitions using the `LinuxParameters` parameter (`devices` and `initProcessEnabled`). For more information, see LinuxParameters.	2 November 2017
Support for Docker flags: cap-add and cap-drop	Added support for Docker's cap-add and cap-drop features in task definitions using the `LinuxParameters` parameter (`capabilities`). For more information, see LinuxParameters.	22 September 2017
Network Load Balancer support	Amazon ECS added support for Network Load Balancers in the Amazon ECS console. For more information, see Creating a Network Load Balancer (p. 387).	7 September 2017

Change	Description	Date
RunTask overrides	Added support for task definition overrides when running a task. This allows you to run a task while changing a task definition without the need to create a new task definition revision. For more information, see Running Tasks (p. 316).	27 June 2017
Amazon ECS scheduled tasks	Added support for scheduling tasks using cron. For more information, see Scheduled Tasks (`cron`) (p. 330).	7 June 2017
Spot Instances in the Amazon ECS console	Added support for creating Spot Fleet container instances within the Amazon ECS console. For more information, see Launching an Amazon ECS Container Instance (p. 219).	6 June 2017
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
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 Subscribing to Amazon ECS-Optimized Amazon Linux AMI Update Notifications (p. 214).	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 Common Use Cases in Amazon ECS (p. 615).	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 Container Instance Draining (p. 239).	24 January 2017
Docker 1.12 support	Added support for Docker 1.12. For more information, see Amazon ECS-optimized AMIs (p. 190).	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 Amazon ECS Task Placement Strategies (p. 321).	29 December 2016
Windows container support in beta	Added support for Windows 2016 containers (beta). For more information, see Windows containers (p. 713).	20 December 2016
Blox OSS support	Added support for Blox OSS, which allows for custom task schedulers. For more information, see Scheduling Amazon ECS Tasks (p. 315).	1 December 2016

Change	Description	Date
Amazon ECS event stream for CloudWatch Events	Amazon ECS now sends container instance and task state changes to CloudWatch Events. For more information, see Amazon ECS Events and EventBridge (p. 428).	21 November 2016
Amazon ECS container logging to CloudWatch Logs	Added support for the awslogs driver to send container log streams to CloudWatch Logs. For more information, see Using the awslogs Log Driver (p. 143).	12 September 2016
Amazon ECS services with Elastic Load Balancing support for dynamic ports	Added support for a load balancer to support multiple instance:port combinations per listener, which increases flexibility for containers. Now you can let Docker dynamically define the container's host port and the ECS scheduler registers the instance:port with the load balancer. For more information, see Service Load Balancing (p. 376).	11 August 2016
IAM roles for Amazon ECS tasks	Added support for associating IAM roles with a task. This provides finer-grained permissions to containers as opposed to a single role for an entire container instance. For more information, see IAM Roles for Tasks (p. 489).	13 July 2016
Amazon ECS CLI support for Docker Compose v2 format	The Amazon ECS CLI added support for Docker Compose v2 format. For more information, see ecs-cli compose (p. 554).	8 July 2016
Docker 1.11 support	Added support for Docker 1.11. For more information, see Amazon ECS-optimized AMIs (p. 190).	31 May 2016
Task automatic scaling	Amazon ECS added support for automatically scaling your tasks run by a service. For more information, see Service Auto Scaling (p. 395).	18 May 2016
Task definition filtering on task family	Added support for filtering a list of task definition based on the task definition family. For more information, see ListTaskDefinitions.	17 May 2016
Docker container and Amazon ECS agent logging	Amazon ECS added ability to send ECS agent and Docker container logs from container instances to CloudWatch Logs to simplify troubleshooting issues.	5 May 2016
Amazon ECS CLI v0.3 released	New version of the Amazon ECS CLI released, which added support for service creation with a load balancer.	11 April 2016
ECS-optimized AMI now supports Amazon Linux 2016.03.	The ECS-optimized AMI added support for Amazon Linux 2016.03. For more information, see Amazon ECS-optimized AMIs (p. 190).	5 April 2016
Docker 1.9 support	Added support for Docker 1.9. For more information, see Amazon ECS-optimized AMIs (p. 190).	22 December 2015
CloudWatch metrics for cluster CPU and memory reservation	Amazon ECS added custom CloudWatch metrics for CPU and memory reservation.	22 December 2015

Change	Description	Date
Amazon ECR	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.	21 December 2015
New Amazon ECS first-run experience	The Amazon ECS console first-run experience added zero-click role creation.	23 November 2015
Task placement across Availability Zones	The Amazon ECS service scheduler added support for task placement across Availability Zones.	8 October 2015
Amazon ECS CLI with support for Docker Compose	The Amazon ECS CLI added support for Docker Compose.	8 October 2015
CloudWatch metrics for Amazon ECS clusters and services	Amazon ECS added custom CloudWatch metrics for CPU and memory utilization for each container instance, service, and task definition family in a cluster. These new metrics can be used to scale container instances in a cluster using Auto Scaling groups or to create custom CloudWatch alarms.	17 August 2015
UDP port support	Added support for UDP ports in task definitions.	7 July 2015
Environment variable overrides	Added support for deregisterTaskDefinition and environment variable overrides for runTask.	18 June 2015
Automated Amazon ECS agent updates	Added ability to see the ECS agent version that is running on a container instance. Also able to update the ECS agent from the AWS Management Console, AWS CLI, and SDK.	11 June 2015
Amazon ECS service scheduler and Elastic Load Balancing integration	Added ability to define a service and associate that service with an Elastic Load Balancing load balancer.	9 April 2015
Amazon ECS GA	Amazon ECS general availability in the US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), and Europe (Ireland) Regions.	9 April 2015

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.