

---

# Amazon ECS

## User Guide for AWS Fargate



## **Amazon ECS: User Guide for AWS Fargate**

Copyright © 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 AWS Fargate? .....	1
Components .....	1
Clusters .....	1
Task definitions .....	1
Tasks .....	1
Services .....	1
Operating System and CPU architecture .....	2
Regions .....	2
Windows containers on AWS Fargate considerations .....	2
Getting started walkthroughs .....	3
Pricing .....	3
Getting started .....	4
Set up .....	4
Sign up for AWS .....	4
Create an IAM user .....	4
Create a virtual private cloud .....	6
Install the AWS CLI .....	7
Creating a container image .....	7
Prerequisites .....	7
Create a Docker image .....	8
Push your image to Amazon Elastic Container Registry .....	10
Clean up .....	11
Next steps .....	11
Using AWS Copilot .....	11
Prerequisites .....	11
Deploy your application using one command .....	12
Deploy your application step by step .....	12
Using the AWS CDK .....	16
Step 1: Set up your AWS CDK project .....	16
Step 2: Use the AWS CDK to define a containerized Web server on Fargate .....	18
Step 3: Test the Web server .....	21
Step 4: Clean up .....	22
Next steps .....	22
Getting started using the classic console .....	23
Using the classic console with Linux containers on AWS Fargate .....	23
Using the classic console with Windows containers on AWS Fargate .....	26
Developer tools overview .....	31
AWS Management Console .....	31
AWS Command Line Interface .....	31
AWS CloudFormation .....	32
AWS Copilot CLI .....	32
AWS CDK .....	32
AWS App2Container .....	33
Amazon ECS CLI .....	33
Docker Desktop integration with Amazon ECS .....	33
AWS SDKs .....	34
Summary .....	34
Using the AWS Copilot CLI .....	35
Installing the AWS Copilot CLI .....	35
Next steps .....	41
Using the Amazon ECS CLI .....	41
Installing the Amazon ECS CLI .....	41
Configuring the Amazon ECS CLI .....	47
Migrating Configuration Files .....	49

Tutorial: Creating a cluster with a Fargate task using the Amazon ECS CLI .....	50
Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI .....	54
Platform versions .....	58
Linux platform versions .....	58
Platform version considerations .....	58
1.4.0 .....	58
1.3.0 .....	59
1.2.0 .....	60
1.1.0 .....	60
1.0.0 .....	61
Migrating to platform version 1.4.0 .....	61
Platform version deprecation .....	61
Windows platform versions .....	62
Platform version considerations .....	62
1.0.0 .....	62
New Amazon Elastic Container Service console .....	63
Cluster management in the new Amazon ECS console .....	63
Creating a cluster for the Fargate launch type using the new console .....	63
Setting the cluster default capacity provider using the new console .....	64
Deleting a cluster using the new console .....	65
Task definition management in the new Amazon ECS console .....	65
Creating a task definition using the new console .....	66
Updating a task definition using the new console .....	69
Deregistering a task definition revision using the new console .....	70
Task management in the new Amazon ECS console .....	70
Run a standalone task using the new console .....	70
Service management in the new Amazon ECS console .....	72
Creating a service using the new console .....	72
Updating a service using the new console .....	73
Deleting a service using the new console .....	74
Clusters .....	76
Cluster concepts .....	76
Creating a cluster using the classic console .....	77
Capacity providers .....	78
Capacity provider concepts .....	78
Capacity provider types .....	78
Capacity provider considerations .....	79
AWS Fargate capacity providers .....	79
Updating cluster settings .....	83
Deleting a cluster using the classic console .....	83
Stopping tasks using the new console .....	84
Task definitions .....	85
Fargate task definition considerations .....	85
Network mode .....	86
Task CPU and memory .....	87
Logging .....	87
Amazon ECS Task execution IAM role .....	87
Example task definition .....	87
Task storage .....	88
Application architecture .....	90
Using the Fargate launch type .....	90
Creating a task definition using the classic console .....	91
Task definition template .....	93
Task definition parameters .....	97
Family .....	97
Launch types .....	97

Task execution role .....	97
Network mode .....	98
Runtime platform .....	98
Task size .....	99
Container definitions .....	100
Proxy configuration .....	119
Volumes .....	121
Tags .....	122
Other task definition parameters .....	123
Launch types .....	124
Fargate launch type .....	124
Working with 64-bit ARM workloads on Amazon ECS .....	125
Considerations .....	126
Specifying the ARM architecture in your task definition .....	126
Interfaces for configuring ARM .....	127
Using data volumes in tasks .....	127
Fargate task storage .....	128
Fargate Linux container platform versions .....	128
Fargate Windows container platform versions .....	128
Amazon EFS volumes .....	129
Bind mounts .....	132
Fargate task networking .....	136
Fargate task networking considerations .....	137
Using a VPC in dual-stack mode .....	138
Using the awslogs log driver .....	138
Turning on the awslogs log driver for your containers .....	139
Creating a log group .....	139
Available awslogs log driver options .....	140
Specifying a log configuration in your task definition .....	141
Viewing awslogs container logs in CloudWatch Logs .....	142
Custom log routing .....	144
Considerations .....	144
Required IAM permissions .....	145
Fluentd buffer limit .....	146
Using Fluent logger libraries or Log4j over TCP .....	147
Using the AWS for Fluent Bit image .....	148
Creating a task definition that uses a FireLens configuration .....	149
Filtering logs using regular expressions .....	152
Concatenate multiline or stack-trace log messages .....	152
Example task definitions .....	169
Private registry authentication for tasks .....	174
Required IAM permissions for private registry authentication .....	174
Enabling private registry authentication .....	175
Specifying sensitive data .....	176
Using Secrets Manager .....	176
Using Systems Manager Parameter Store .....	182
Specifying environment variables .....	186
Considerations for specifying environment variable files .....	187
Required IAM permissions .....	188
Example task definitions .....	189
Example: Webserver .....	189
Example: splunk log driver .....	191
Example: fluentd log driver .....	191
Example: gelf log driver .....	192
Example: Container dependency .....	192
Windows sample task definitions .....	193
Updating a task definition using the classic console .....	194

Deregistering a task definition revision .....	194
Account settings .....	196
Amazon Resource Names (ARNs) and IDs .....	197
ARN and resource ID format timeline .....	198
Viewing account settings .....	198
Modifying account settings .....	199
Scheduling tasks .....	201
Run a standalone task .....	202
Scheduled tasks .....	204
Create a scheduled task .....	204
View your scheduled tasks .....	208
Edit a scheduled task .....	208
Task maintenance .....	208
Understanding the task retirement notice .....	210
Services .....	212
Service scheduler concepts .....	212
Replica .....	213
Additional service concepts .....	213
Service definition parameters .....	213
Launch type .....	213
Capacity provider strategy .....	214
Task definition .....	215
Platform operating system .....	215
Platform version .....	215
Cluster .....	216
Service name .....	216
Scheduling strategy .....	216
Desired count .....	217
Deployment configuration .....	217
Deployment controller .....	218
Task placement .....	219
Tags .....	220
Network configuration .....	221
Client token .....	224
Service definition template .....	224
Creating a service .....	226
Creating a service using the classic Amazon ECS console .....	226
Updating a service .....	237
Updating a service using the classic console .....	238
Deleting a service .....	239
Deployment types .....	240
Rolling update .....	240
Blue/Green deployment with CodeDeploy .....	243
External deployment .....	247
Service load balancing .....	252
Service load balancing considerations .....	253
Load balancer types .....	254
Creating a load balancer .....	256
Registering multiple target groups with a service .....	261
Service auto scaling .....	263
Service auto scaling and deployments .....	263
IAM permissions required for service auto scaling .....	264
Target tracking scaling policies .....	265
Step scaling policies .....	269
Service Discovery .....	271
Service Discovery concepts .....	272
Service discovery considerations .....	273

Amazon ECS classic console experience .....	274
Service discovery pricing .....	274
Service throttle logic .....	274
Resources and tags .....	276
Tagging your resources .....	276
Tag basics .....	276
Tagging your resources .....	277
Tag restrictions .....	278
Tagging your resources for billing .....	278
Working with tags using the console .....	279
Working with tags using the CLI or API .....	280
Service quotas .....	281
Amazon ECS service quotas .....	282
AWS Fargate service quotas .....	284
Managing your Amazon ECS and AWS Fargate service quotas in the AWS Management Console ..	284
AWS Fargate Regions .....	285
Supported Regions for Linux containers on AWS Fargate .....	285
Supported Regions for Windows containers on AWS Fargate .....	286
Usage Reports .....	287
Monitoring .....	289
Monitoring tools .....	289
Automated Tools .....	289
Manual Tools .....	290
CloudWatch metrics .....	290
Enabling CloudWatch metrics .....	291
Available metrics and dimensions .....	291
Service utilization .....	293
Service RUNNING task count .....	294
Viewing Amazon ECS metrics .....	295
Usage metrics .....	296
Creating a CloudWatch alarm to monitor Fargate resource usage metrics .....	297
Events and EventBridge .....	297
Amazon ECS events .....	298
Handling events .....	306
CloudWatch Container Insights .....	308
Container Insights considerations .....	308
Setting up CloudWatch Container Insights for cluster and service level metrics .....	308
Collecting application trace data .....	310
Required IAM permissions for AWS Distro for OpenTelemetry integration with AWS X-Ray .....	310
Specifying the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration in your task definition .....	311
Collecting application metrics .....	312
Exporting application metrics to Amazon CloudWatch .....	312
Exporting application metrics to Amazon Managed Service for Prometheus .....	315
Logging Amazon ECS API calls with AWS CloudTrail .....	317
Amazon ECS information in CloudTrail .....	317
Understanding Amazon ECS log file entries .....	318
Security .....	320
Identity and Access Management .....	320
Audience .....	321
Authenticating with identities .....	321
Managing access using policies .....	323
How Amazon Elastic Container Service works with IAM .....	325
Identity-based policy examples .....	331
AWS managed policies for Amazon ECS .....	340
Service-linked role .....	348
Task execution IAM role .....	354

ECS Anywhere IAM role .....	358
IAM roles for tasks .....	360
CodeDeploy IAM Role .....	365
CloudWatch Events IAM Role .....	368
Additional configuration for Windows IAM roles for tasks .....	371
Troubleshooting .....	373
Logging and Monitoring .....	375
Compliance Validation .....	376
Infrastructure Security .....	376
Interface VPC endpoints (AWS PrivateLink) .....	377
Task metadata endpoint .....	380
Task metadata endpoint v4 .....	380
Enabling the task metadata endpoint .....	380
Task metadata endpoint version 4 paths .....	380
Task metadata JSON response .....	381
Examples .....	383
Task metadata endpoint v3 .....	389
Enabling Task Metadata .....	389
Task Metadata Endpoint Paths .....	389
Task Metadata JSON Response .....	389
Example Task Metadata Response .....	392
Working with other services .....	394
Using Amazon ECR with Amazon ECS .....	394
Using Amazon ECR Images with Amazon ECS .....	394
Creating Amazon ECS resources with AWS CloudFormation .....	395
Amazon ECS and AWS CloudFormation templates .....	395
Learn more about AWS CloudFormation .....	395
Use App Mesh with Amazon ECS .....	395
Tutorials .....	397
Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI .....	397
Prerequisites .....	397
Step 1: Create a Cluster .....	398
Step 2: Register a Linux Task Definition .....	398
Step 3: List Task Definitions .....	399
Step 4: Create a Service .....	399
Step 5: List Services .....	400
Step 6: Describe the Running Service .....	400
Step 7: Test .....	402
Step 8: Clean Up .....	403
Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI .....	403
Prerequisites .....	403
Step 1: Create a Cluster .....	404
Step 2: Register a Windows Task Definition .....	404
Step 3: List task definitions .....	405
Step 4: Create a service .....	406
Step 5: List services .....	406
Step 6: Describe the Running Service .....	406
Step 7: Clean Up .....	408
Tutorial: Specifying sensitive data using Secrets Manager secrets .....	408
Prerequisites .....	409
Step 1: Create an Secrets Manager secret .....	409
Step 2: Update your task execution IAM role .....	409
Step 3: Create an Amazon ECS task definition .....	410
Step 4: Create an Amazon ECS cluster .....	411
Step 5: Run an Amazon ECS task .....	412
Step 6: Verify .....	412
Step 7: Clean up .....	413



Tutorial: Creating a service using Service Discovery .....	413
Prerequisites .....	414
Step 1: Create the Service Discovery resources in AWS Cloud Map .....	414
Step 2: Create the Amazon ECS resources .....	415
Step 3: Verify Service Discovery in AWS Cloud Map .....	417
Step 4: Clean up .....	418
Tutorial: Creating a service using a blue/green deployment .....	420
Prerequisites .....	420
Step 1: Create an Application Load Balancer .....	420
Step 2: Create an Amazon ECS cluster .....	421
Step 3: Register a task definition .....	422
Step 4: Create an Amazon ECS service .....	422
Step 5: Create the AWS CodeDeploy resources .....	423
Step 6: Create and monitor a CodeDeploy deployment .....	425
Step 7: Clean up .....	427
Tutorial: Listening for Amazon ECS CloudWatch Events .....	429
Prerequisite: Set up a test cluster .....	429
Step 1: Create the Lambda function .....	429
Step 2: Register an event rule .....	429
Step 3: Test your rule .....	430
Tutorial: Sending Amazon Simple Notification Service alerts for task stopped events .....	430
Prerequisite: Set up a test cluster .....	430
Step 1: Create and subscribe to an Amazon SNS topic .....	430
Step 2: Register an event rule .....	431
Step 3: Test your rule .....	432
Troubleshooting .....	433
Using Amazon ECS Exec for debugging .....	433
Architecture .....	433
Considerations for using ECS Exec .....	434
Prerequisites for using ECS Exec .....	434
Enabling and using ECS Exec .....	435
Logging and Auditing using ECS Exec .....	437
Using IAM policies to limit access to ECS Exec .....	439
Troubleshooting issues with ECS Exec .....	442
Checking stopped tasks for errors .....	442
Additional resources .....	444
Stopped tasks error codes .....	444
CannotPullContainer task errors .....	447
Service event messages .....	450
Service event messages .....	452
Invalid CPU or memory value specified .....	453
Troubleshooting service load balancers .....	454
Troubleshooting service auto scaling .....	455
AWS Fargate throttling quotas .....	455
Throttling the RunTask API .....	456
Adjusting rate quotas .....	456
API failure reasons .....	456
Document history .....	458
AWS glossary .....	470

# What is 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 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 Amazon ECS tasks and services with the Fargate launch type or a Fargate capacity provider, you package your application in containers, specify the Operating System, 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.

For information about Fargate architecture, see [Using the Fargate launch type](#) in the Amazon Elastic Container Service Developer Guide

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

## Components

### Clusters

An Amazon ECS *cluster* is a logical grouping of tasks or services. You can use clusters to isolate your applications. When your tasks are run on Fargate, your cluster resources are also managed by Fargate.

### Task definitions

A *task definition* is a text file that describes one or more containers that form your application. It's in JSON format. You can use it to describe up to a maximum of ten containers. The task definition functions as a blueprint for your application. It specifies the various parameters for your application. For example, you can use it to specify parameters for the operating system, which containers to use, which ports to open for your application, and what data volumes to use with the containers in the task. The specific parameters available for your task definition depend on the needs of your specific application.

Your entire application stack doesn't need to be on a single task definition. In fact, we recommend spanning your application across multiple task definitions. You can do this by combining related containers into their own task definitions, each representing a single component.

### Tasks

A *task* is the instantiation of a task definition within a cluster. After you create a task definition for your application within Amazon ECS, you can specify the number of tasks to run on your cluster. You can run a standalone task, or you can run a task as part of a service.

### Services

You can use an Amazon ECS *service* to run and maintain your desired number of tasks simultaneously in an Amazon ECS cluster. How it works is that, if any of your tasks fail or stop for any reason, the Amazon

ECS service scheduler launches another instance based on your task definition. It does this to replace it and thereby maintain your desired number of tasks in the service.

## Operating System and CPU architecture

The following operating systems are supported:

- Amazon Linux 2
- Windows Server 2019 Full
- Windows Server 2019 Core

If you use Windows containers on Fargate, review [the section called “Windows containers on AWS Fargate considerations” \(p. 2\)](#).

There are 2 architectures available for the Amazon ECS task definition, ARM and X86\_64.

When you run Windows containers on Fargate, you must have an X86\_64 CPU architecture.

When you run Linux containers on Fargate, you can use the X86\_64 CPU architecture, or the ARM64 architecture for your ARM-based applications. For more information, see [the section called “Working with 64-bit ARM workloads on Amazon ECS” \(p. 125\)](#).

## Regions

For information about the Regions that support Linux containers on Fargate, see [the section called “Supported Regions for Linux containers on AWS Fargate” \(p. 285\)](#).

For information about the Regions that support Windows containers on Fargate, see [the section called “Supported Regions for Windows containers on AWS Fargate” \(p. 286\)](#).

## Windows containers on AWS Fargate considerations

Windows containers on AWS Fargate supports the following operating systems:

- Windows Server 2019 Full
- Windows Server 2019 Core

AWS handles the operating system license management, so you do not need any additional Microsoft licenses.

Windows containers on AWS Fargate supports the awslogs driver. For more information, see [the section called “Using the awslogs log driver” \(p. 138\)](#).

Your tasks can run either Linux containers or Windows containers. If you need run both container types, you must create separate tasks.

The following features are not supported on Windows containers on Fargate:

- Group managed service accounts (gMSA)

- Amazon FSx
- ENI trunking
- App Mesh service and proxy integration for tasks
- Firelens log router integration for tasks
- Configurable ephemeral storage
- EFS volumes
- The Fargate Spot capacity provider
- Image volumes

The Dockerfile `volume` option is ignored. Instead, use bind mounts in your task definition. For more information, see [Bind mounts](#) (p. 132).

## Getting started walkthroughs

The following walkthroughs help you get started using Amazon ECS on Fargate.

- [Getting started with the classic console using Linux containers on AWS Fargate](#) (p. 23)
- [the section called “Using the classic console with Windows containers on AWS Fargate”](#) (p. 26)
- [Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI](#) (p. 397)
- [the section called “Using the classic console with Windows containers on AWS Fargate”](#) (p. 26)
- [the section called “Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI”](#) (p. 403)

For more information about Amazon Elastic Container Service, see [What is Amazon ECS?](#).

## Pricing

With Amazon ECS on AWS Fargate, you pay for the vCPU and memory resources your tasks use. For more information, see [Fargate Pricing](#).

Fargate also offers Savings Plans which provides significant savings on your AWS usage. For more information, see the [Savings Plans User Guide](#).

To see your bill, go to the **Billing and Cost Management Dashboard** in the [AWS Billing and Cost Management console](#). Your bill contains links to usage reports that provide details about your bill. To learn more about AWS account billing, see [AWS Account Billing](#).

If you have questions concerning AWS billing, accounts, and events, [contact AWS Support](#).

For an overview of Trusted Advisor, a service that helps you optimize the costs, security, and performance of your AWS environment, see [AWS Trusted Advisor](#).

# Getting started with Amazon ECS

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

## Contents

- [Set up to use Amazon ECS \(p. 4\)](#)
- [Creating a container image for use on Amazon ECS \(p. 7\)](#)
- [Getting started with Amazon ECS using AWS Copilot \(p. 11\)](#)
- [Getting started with Amazon ECS using the AWS CDK \(p. 16\)](#)
- [Getting started with Amazon ECS using the classic console \(p. 23\)](#)

## Set up to use Amazon ECS

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

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

## 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 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. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

#### Note

We strongly recommend that you adhere to the best practice of using the **Administrator** IAM user that follows 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 users**.
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 top of the IAM dashboard, to the right of your sign-in link, choose **Customize** 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 virtual private cloud

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined.

### 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, [Install the AWS CLI \(p. 7\)](#). 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.

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

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

For more information about Amazon VPC, see [What is Amazon VPC?](#) in the *Amazon VPC User Guide*.

## 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*.

## Creating a container image for use on Amazon ECS

Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see [Deploying Docker containers on Amazon ECS](#).

The purpose of the steps outlined here is to walk you through creating your first Docker image and pushing that image to Amazon ECR, which is a container registry, for use in your Amazon ECS task definitions. This walkthrough assumes that you possess a basic understanding of what Docker is and how it works. For more information about Docker, see [What is Docker?](#) and the [Docker overview](#).

### Important

AWS and Docker have collaborated to make a simplified developer experience that enables you to deploy and manage containers on Amazon ECS directly using Docker tools. You can now build and test your containers locally using Docker Desktop and Docker Compose, and then deploy them to Amazon ECS on Fargate. To get started with the Amazon ECS and Docker integration, download Docker Desktop and optionally sign up for a Docker ID. For more information, see [Docker Desktop](#) and [Docker ID signup](#).

## Prerequisites

Before you begin, ensure the following prerequisites are met.

- Ensure you have completed the Amazon ECR setup steps. For more information, see [Setting up for Amazon ECR](#) in the *Amazon Elastic Container Registry User Guide*.
- Your user has the required IAM permissions to access and use the Amazon ECR service. For more information, see [Amazon ECR managed policies](#).
- You have Docker installed. For Docker installation steps for Amazon Linux 2, see [Installing Docker on Amazon Linux 2 \(p. 7\)](#). For all other operating systems, see the Docker documentation at [Docker Desktop overview](#).
- You have the AWS CLI installed and configured. For more information, see [Installing the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

If you don't have or need a local development environment and you prefer to use an Amazon EC2 instance to use Docker, we provide the following steps to launch an Amazon EC2 instance using Amazon Linux 2 and install Docker Engine and the Docker CLI.

### Installing Docker on Amazon Linux 2

Docker Desktop is an easy-to-install application for your Mac or Windows environment that enables you to build and share containerized applications and microservices. Docker Desktop includes Docker Engine,



the Docker CLI client, Docker Compose, and other tools that are helpful when using Docker with Amazon ECS. For more information about how to install Docker Desktop on your preferred operating system, see [Docker Desktop overview](#).

### 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 Engine package.

```
sudo amazon-linux-extras install docker
```

#### Important

This step assumes you are using the Amazon Linux 2 AMI for your instance. For all other operating systems, see [Docker Desktop overview](#).

5. Start the Docker service.

```
sudo service docker start
```

(Optional) To ensure that the Docker daemon starts after each system reboot, run the following command:

```
sudo systemctl enable docker
```

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 you 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 the Amazon ECR container registry so you can use it in an Amazon ECS task definition.

## To create a Docker image of a simple web application

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

```
touch Dockerfile
```

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

```
FROM 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. You can use the Docker CLI to push, pull, and manage images in your Amazon ECR repositories. For Amazon ECR product details, featured customer case studies, and FAQs, see the [Amazon Elastic Container Registry product detail pages](#).

### To tag your image and push it to Amazon ECR

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

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

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

Output:

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

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

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

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

```
aws ecr get-login-password | 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

To continue on with creating an Amazon ECS task definition and launching a task with your container image, skip to the [Next steps](#) (p. 11). When you are done experimenting with your Amazon ECR image, you can delete the repository so you are not charged for image storage.

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

## Next steps

After you have created and pushed your container image to Amazon ECR, you should consider the following next steps.

- [Getting started with Amazon ECS using the classic console](#) (p. 23)
- [Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI](#) (p. 397)

# Getting started with Amazon ECS using AWS Copilot

Get started with Amazon ECS using AWS Copilot by deploying an Amazon ECS application.

## Prerequisites

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

- Set up an AWS account. For more information see [Set up to use Amazon ECS](#) (p. 4).
- Install the AWS Copilot CLI. Releases currently support Linux and macOS systems. For more information, see [Installing the AWS Copilot CLI](#) (p. 35).

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

## Deploy your application using one command

Make sure that you have the AWS command line tool installed and have already run `aws configure` before you start.

Deploy the application using the following command.

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

## Deploy your application step by step

### Step 1: Configure your credentials

Run `aws configure` to set up a default profile that the AWS Copilot CLI uses to manage your application and services.

```
aws configure
```

### Step 2: Clone the demo app

Clone a simple Flask application and Dockerfile.

```
git clone https://github.com/aws-samples/amazon-ecs-cli-sample-app.git demo-app
```

### Step 3: Set up your application

1. From within the `demo-app` directory, run the `init` command.

```
copilot init
```

AWS Copilot walks you through the setup of your **first application and service** with a series of terminal prompts, starting with **next step**. If you have already used AWS Copilot to deploy applications, you're prompted to choose one from a list of application names.

2. Name your application.

```
What would you like to name your application? [? for help]
```

Enter **demo**.

## Step 4: Set up an ECS Service in your "demo" Application

1. You're prompted to choose a service type. You're building a simple Flask application that serves a small API.

```
Which service type best represents your service's architecture? [Use arrows to move,
type to filter, ? for more help]
> Load Balanced Web Service
  Backend Service
  Scheduled Job
```

Choose **Load Balanced Web Service**.

2. Provide a name for your service.

```
What do you want to name this Load Balanced Web Service? [? for help]
```

Enter **api** for your service name.

3. Select a Dockerfile.

```
Which Dockerfile would you like to use for api? [Use arrows to move, type to filter, ?
for more help]
> ./Dockerfile
  Use an existing image instead
```

Choose **Dockerfile**.

4. Define port.

```
Which port do you want customer traffic sent to? [? for help] (80)
```

Enter **80** or accept default.

5. You will see a log showing the application resources being created.

```
Creating the infrastructure to manage services under application demo.
```

6. After the application resources are created, deploy a test environment.

```
Would you like to deploy a test environment? [? for help] (y/N)
```

Enter **y**.

```
Proposing infrastructure changes for the test environment.
```

7. You will see a log displaying the status of your application deployment.

```
Note: It's best to run this command in the root of your Git repository.
Welcome to the Copilot CLI! We're going to walk you through some questions
to help you get set up with an application on ECS. An application is a collection of
containerized services that operate together.
```

```
Use existing application: No
Application name: demo
Workload type: Load Balanced Web Service
Service name: api
Dockerfile: ./Dockerfile
```

```
no EXPOSE statements in Dockerfile ./Dockerfile
Port: 80
Ok great, we'll set up a Load Balanced Web Service named api in application demo
  listening on port 80.

# Created the infrastructure to manage services under application demo.

# Wrote the manifest for service api at copilot/api/manifest.yml
Your manifest contains configurations like your container size and port (:80).

# Created ECR repositories for service api.

All right, you're all set for local development.
Deploy: Yes

# Created the infrastructure for the test environment.
- Virtual private cloud on 2 availability zones to hold your services      [Complete]
- Virtual private cloud on 2 availability zones to hold your services      [Complete]
  - Internet gateway to connect the network to the internet                [Complete]
  - Public subnets for internet facing services                           [Complete]
  - Private subnets for services that can't be reached from the internet  [Complete]
  - Routing tables for services to talk with each other                    [Complete]
- ECS Cluster to hold your services                                       [Complete]
# Linked account aws_account_id and region region to application demo.

# Created environment test in region region under application demo.

Environment test is already on the latest version v1.0.0, skip upgrade.
[+] Building 0.8s (7/7) FINISHED
=> [internal] load .dockerignore
    0.1s
=> => transferring context: 2B
    0.0s
=> [internal] load build definition from Dockerfile
    0.0s
=> => transferring dockerfile: 37B
    0.0s
=> [internal] load metadata for docker.io/library/nginx:latest
    0.7s
=> [internal] load build context
    0.0s
=> => transferring context: 32B
    0.0s
=> [1/2] FROM docker.io/library/
nginx@sha256:aeade65e99e5d5e7ce162833636f692354c227ff438556e5f3ed0335b7cc2f1b    0.0s
=> CACHED [2/2] COPY index.html /usr/share/nginx/html
    0.0s
=> exporting to image
    0.0s
=> => exporting layers
    0.0s
=> => writing image
sha256:3ee02fd4c0f67d7bd808ed7fc73263880649834cbb05d5ca62380f539f4884c4
    0.0s
=> => naming to aws_account_id.dkr.ecr.region.amazonaws.com/demo/api:cee7709
    0.0s
WARNING! Your password will be stored unencrypted in /home/user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
The push refers to repository [aws_account_id.dkr.ecr.region.amazonaws.com/demo/api]
592a5c0c47f1: Pushed
6c7de695ede3: Pushed
2f4accd375d9: Pushed
ffc9b21953f4: Pushed
```

```
cee7709: digest: sha_digest  
  
# Deployed api, you can access it at http://demo-  
Publi-10Q8VMS2VC2WG-561733989.region.elb.amazonaws.com.
```

## Step 5: Verify your application is running

View the status of your application by using the following commands.

List all of your AWS Copilot applications.

```
copilot app ls
```

Show information about the environments and services in your application.

```
copilot app show
```

Show information about your environments.

```
copilot env ls
```

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

```
copilot svc show
```

List of all the services in an application.

```
copilot svc ls
```

Show logs of a deployed service.

```
copilot svc logs
```

Show service status.

```
copilot svc status
```

List available commands and options.

```
copilot --help
```

```
copilot init --help
```

## Step 6. Learn to create a CI/CD Pipeline

Instructions can be found in the [ECS Workshop](#) detailing how to fully automate a CI/CD pipeline and git workflow using AWS Copilot.

## Step 7: Clean up

Run the following command to delete and clean up all resources.



```
copilot app delete
```

## Getting started with Amazon ECS using the AWS CDK

This topic shows you how to deploy a containerized Web server with Amazon Elastic Container Service and the AWS Cloud Development Kit (CDK) on Fargate. The AWS CDK is an Infrastructure as Code (IAC) framework that lets you define AWS infrastructure using a full-fledged programming language. You write an app in one of the CDK's supported languages, containing one or more stacks, then synthesize it to an AWS CloudFormation template and deploy the resources to your AWS account.

The AWS Construct Library, included with the CDK, provides APIs that model the resources provided by every AWS service. For the most popular services, the library provides curated constructs that provide smart defaults and implement best practices with fewer required parameters. One of these modules, [aws-ecs-patterns](#), provides high-level abstractions that let you define your containerized service and all necessary supporting resources in only a few lines of code.

The construct we'll be using in this topic is [ApplicationLoadBalancedFargateService](#). As you can likely tell from the name, this construct deploys an Amazon ECS service on Fargate behind an application load balancer. The `aws-ecs-patterns` module also includes constructs that use a network load balancer and/or run on Amazon EC2, if you'd prefer those options.

Before embarking on this task, set up your AWS CDK development environment as described in [Getting Started With the AWS CDK - Prerequisites](#), then install the AWS CDK by issuing:

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

### Note

These instructions assume you are using AWS CDK v2.

### Topics

- [Step 1: Set up your AWS CDK project \(p. 16\)](#)
- [Step 2: Use the AWS CDK to define a containerized Web server on Fargate \(p. 18\)](#)
- [Step 3: Test the Web server \(p. 21\)](#)
- [Step 4: Clean up \(p. 22\)](#)
- [Next steps \(p. 22\)](#)

## Step 1: Set up your AWS CDK project

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

### TypeScript

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

### JavaScript

```
mkdir hello-ecs
```

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

## Python

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

After the project has been initialized, activate the project's virtual environment and install the AWS CDK's baseline dependencies.

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

## Java

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

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

## C#

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

### Note

Be sure to name the directory `hello-ecs` as shown. The AWS CDK application template uses the name of the project directory to generate names for source files and classes. If you use a different name, your app will not match these instructions.

AWS CDK v2 includes stable constructs for all AWS services in a single package, dubbed `aws-cdk-lib`. This package is installed as a dependency when you initialize the project (or, in some languages, the first time you build it). In this topic, we use an Amazon ECS Patterns construct, which provides high-level abstractions for working with Amazon ECS. In turn, this module relies on Amazon ECS constructs and others to provision the resources needed by your Amazon ECS application.

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

## TypeScript

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

## JavaScript

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

#### Python

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

#### Java

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

#### C#

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

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

We'll use the container image [amazon-ecs-sample](#) from DockerHub. This image contains a PHP Web app running under Amazon Linux 2.

In the AWS CDK project you created, edit the file containing the definition of the stack to look like the code below. You'll recognize the instantiation of the `ApplicationLoadBalancedFargateService` construct—or at least its name.

#### Note

What's a stack? The stack is the unit of deployment: all resources must be in a stack, and all the resources in a stack are deployed together. If a resource fails to deploy, any other resources already deployed are rolled back. An AWS CDK app can contain multiple stacks, and resources in one stack can refer to resources in another.

#### TypeScript

Update `lib/hello-ecs-stack.ts` to read as follows.

```
import * as cdk from '@aws-cdk-lib';
import { Construct } from 'constructs';

import * as ecs from '@aws-cdk-lib/aws-ecs';
import * as ecsp from '@aws-cdk-lib/aws-ecs-patterns';

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

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

#### JavaScript

Update `lib/hello-ecs-stack.js` to read as follows.

```
const cdk = require('@aws-cdk-lib');
const { Construct } = require('constructs');

const ecs = require('@aws-cdk-lib/aws-ecs');
const ecsp = require('@aws-cdk-lib/aws-ecs-patterns');

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

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

module.exports = { HelloEcsStack }
```

## Python

Update `hello-ecs/hello_ecs_stack.py` to read as follows.

```
import aws_cdk as cdk
from constructs import Construct

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

class HelloEcsStack(cdk.Stack):

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

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

## Java

Update `src/main/java/com.myorg/HelloEcsStack.java` to read as follows.

```
package com.myorg;

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

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

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

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

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

## C#

Update `src/HelloEcs/HelloEcsStack.cs` to read as follows.

```
using Amazon.CDK;
using Constructs;

using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;

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

You can see in this short snippet:

- The service's logical name, `MyWebServer`.
- The container image, obtained from DockerHub, `amazon/amazon-ecs-sample`.
- The fact that the load balancer will have a public address and will thus be accessible from the Internet.

If you omit, as we have done here, the Amazon ECS cluster, the underlying Amazon Virtual Private Cloud and Amazon EC2 instances, an Auto Scaling Group, the Application Load Balancer, the necessary IAM roles and policies, and other AWS resources required to deploy the Web server, the AWS CDK will also create these resources. Some automatically-provisioned resources will be shared by all Amazon ECS services defined in the stack.

Save the source file, then issue `cdk synth` in the app's main directory. The AWS CDK runs the app and synthesizes an AWS CloudFormation template from it, then displays the template. The template is about 600 lines of YAML, so only the beginning is shown here. (Your template may have differences from ours.)

```
Resources:
  MyWebServerLB3B5FD3AB:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      LoadBalancerAttributes:
        - Key: deletion_protection.enabled
          Value: "false"
      Scheme: internet-facing
      SecurityGroups:
        - Fn::GetAtt:
            - MyWebServerLBSecurityGroup01B285AA
            - GroupId
      Subnets:
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1Subnet3C273B99
        - Ref: EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2Subnet95FF715A
      Type: application
    DependsOn:
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet1DefaultRouteFF4E2178
      - EcsDefaultClusterMnL3mNNYNVpcPublicSubnet2DefaultRouteB1375520
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/Resource
  MyWebServerLBSecurityGroup01B285AA:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Automatically created Security Group for ELB
  HelloEcsStackMyWebServerLB06757F57:
    SecurityGroupIngress:
      - CidrIp: 0.0.0.0/0
        Description: Allow from anyone on port 80
        FromPort: 80
        IpProtocol: tcp
        ToPort: 80
    VpcId:
      Ref: EcsDefaultClusterMnL3mNNYNVpc7788A521
    Metadata:
      aws:cdk:path: HelloEcsStack/MyWebServer/LB/SecurityGroup/Resource
# and so on for another few hundred lines
```

To actually deploy the service in your AWS account, issue `cdk deploy`. You'll be asked to approve the IAM policies the AWS CDK has generated.

Deployment will take several minutes. You'll see the AWS CDK create quite a number of resources. The last few lines of the output from the deployment include the public hostname of the load balancer and an HTTP URL for your new Web server.

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

## Step 3: Test the Web server

Copy the URL from the deployment output and paste it into your Web browser. You should see a welcome message from the Web server.

# Simple PHP App

## Congratulations

Your PHP application is now running on a container in Amazon ECS.

The container is running PHP version 5.4.16.

## Step 4: Clean up

Now that you're done with the Web server (it doesn't do anything besides display the Congratulations message), you can tear down the service using the CDK. Issue `cdk destroy` in your app's main directory. Doing this will prevent unintended AWS charges.

## Next steps

To learn more about developing AWS infrastructure using the AWS CDK, see the [AWS CDK Developer Guide](#).

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

TypeScript

[Working with the AWS CDK in TypeScript](#)

JavaScript

[Working with the AWS CDK in JavaScript](#)

Python

[Working with the AWS CDK in Python](#)

Java

[Working with the AWS CDK in Java](#)

C#

[Working with the AWS CDK in C#](#)

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

- [aws-ecs](#)
- [aws-ecs-patterns](#)

# Getting started with Amazon ECS using the classic console

The following guides provide an introduction to the classic AWS Management Console to complete the common tasks to run your containers on Amazon ECS and AWS Fargate.

## Contents

- [Getting started with the classic console using Linux containers on AWS Fargate \(p. 23\)](#)
- [Getting started with the classic Amazon ECS console using Windows containers on AWS Fargate \(p. 26\)](#)

## Getting started with the classic console using Linux containers on AWS 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 on Fargate, see [What is AWS Fargate? \(p. 1\)](#).

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 classic 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.

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 [Set up to use Amazon ECS \(p. 4\)](#) and that your AWS user has either the permissions specified in the `AdministratorAccess` or [Amazon ECS first-run wizard permissions \(p. 333\)](#) 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 [Set up to use Amazon ECS \(p. 4\)](#).
- 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 \(p. 354\)](#).

## 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 classic 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 [Fargate launch type \(p. 124\)](#).

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 \(p. 100\)](#).

**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 \(p. 97\)](#).

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 \(p. 252\)](#).

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.

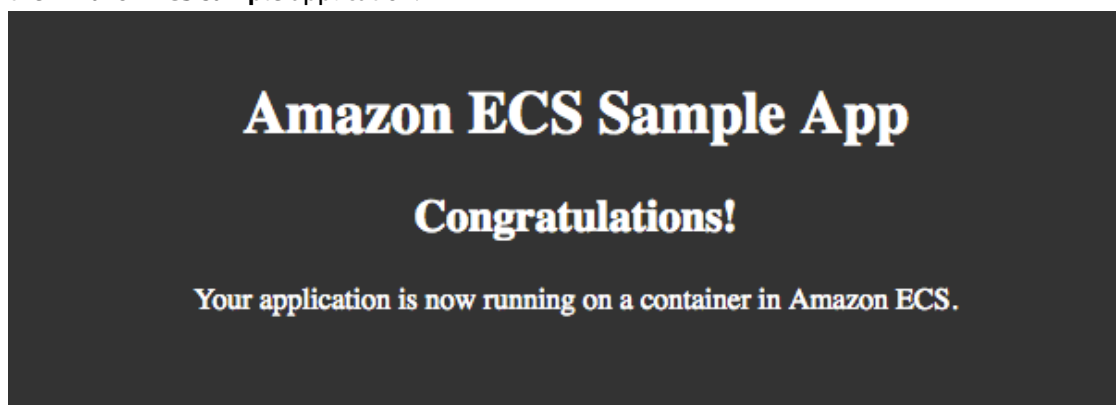
## Step 4: Review

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.

## Getting started with the classic Amazon ECS console using Windows containers on AWS 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 on Fargate, see [What is AWS Fargate? \(p. 1\)](#).

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.

### Prerequisites

Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 4\)](#) and that your AWS user has either the permissions specified in the `AdministratorAccess` or [Amazon ECS first-run wizard permissions \(p. 333\)](#) 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 [Set up to use Amazon ECS \(p. 4\)](#).
- 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 \(p. 354\)](#).

## Step 1: Create a cluster

You can create a new cluster called `windows`.

### 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 **Networking only** 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 **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. To create a new VPC, select **Create VPC**.
  - b. (Optional) If you chose to create a new VPC, for **CIDR Block**, enter a CIDR block for your VPC. For more information, see [Your VPC and Subnets](#) in the *Amazon VPC User Guide*.
  - c. For **Subnet 1** and **Subnet 2**, enter the CIDR range for each subnet.
7. 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](#).
8. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 308\)](#).
9. 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 `mcr.microsoft.com/windows/servercore/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 **Fargate, Next step**.
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**.

Use one of the following for `operatingSystemFamily`:

- `WINDOWS_SERVER_2019_FULL`
- `WINDOWS_SERVER_2019_CORE`

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

```
    ],
    "entryPoint": [
      "powershell",
      "-Command"
    ],
    "essential": true,
    "cpu": 2048,
    "memory": 4096,
    "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-
ltsc2019",
    "logConfiguration": {
      "logDriver": "awslogs",
      "options": {
        "awslogs-group": "/ecs/fargate-windows-task-definition",
        "awslogs-region": "us-east-1",
        "awslogs-stream-prefix": "ecs"
      }
    },
    "name": "sample_windows_app",
    "portMappings": [
      {
        "hostPort": 80,
        "containerPort": 80,
        "protocol": "tcp"
      }
    ]
  }
},
"memory": "4096",
"cpu": "2048",
"networkMode": "awsvpc",
"family": "windows-simple-iis-2019-core",
"executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
"runtimePlatform": {
  "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"
},
"requiresCompatibilities": [
  "FARGATE"
]
}
```

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

### To register the sample task definition with the AWS CLI

1. Create a file called `windows_fargate_sample_app.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_fargate_sample_app.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\_fargate\_sample\_app** registration confirmation page, choose **Actions, Create Service**.
2. On the **Create Service** page, enter the following information and then choose **Create service**.
  - **Launch type:** Fargate
  - **Platform operating system:** WINDOWS\_SERVER\_2019\_FULL or WINDOWS\_SERVER\_2019\_CORE
  - **Cluster:** windows
  - **Service name:** windows\_fargate\_sample\_app
  - **Service type:** REPLICAS
  - **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_fargate_sample_app
```

## 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\_fargate\_sample\_app** service.
4. On the **Service: windows\_fargate\_sample\_app** page, choose the task ID for the task in your service.
5. On the **Task** page, expand the windows\_fargate 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.

**Amazon ECS Sample App**

**Congratulations!**

Your application is now running on a container in Amazon ECS.

## Step 5: Clean Up

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

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

1. 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 developer tools overview

Whether you are part of a large enterprise or a startup, Amazon ECS offers a variety of tools that can help you to get your containers up and running quickly, regardless of your level of expertise. You can work with Amazon ECS in the following ways.

- Learn about, develop, manage and visualize your container applications and services using the [AWS Management Console \(p. 31\)](#).
- Perform specific actions to Amazon ECS resources with automated deployments through programming or scripts using the [AWS Command Line Interface \(p. 31\)](#), [AWS SDKs \(p. 34\)](#) or the ECS API.
- Define and manage all AWS resources in your environment with automated deployment using [AWS CloudFormation \(p. 32\)](#).
- Use the complete [AWS Copilot CLI \(p. 32\)](#) end-to-end developer workflow to create, release, and operate container applications that comply with AWS best practices for infrastructure.
- Using your preferred programming language, define infrastructure or architecture as code with the [AWS CDK \(p. 32\)](#).
- Containerize applications that are hosted on premises or on Amazon EC2 instances or both by using the [AWS App2Container \(p. 33\)](#) integrated portability and tooling ecosystem for containers.
- Deploy a Docker Compose application to Amazon ECS or test local containers with containers running in ECS, using the [Amazon ECS CLI \(p. 33\)](#).
- Launch containers from [Docker Desktop integration with Amazon ECS \(p. 33\)](#) using Amazon ECS in Docker Desktop.

## AWS Management Console

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

For a tutorial that guides you through the console, see [Getting started with Amazon ECS \(p. 4\)](#).

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

Start with the AWS Management Console.

## AWS Command Line Interface

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



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](#).

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

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

## AWS CloudFormation

[AWS CloudFormation](#) and [Terraform](#) for Amazon ECS both provide powerful ways for you to define your infrastructure as code. You can easily track which version of your template or AWS CloudFormation stack is running at any time and rollback to a previous version if needed. You can perform infrastructure and application deployments in the same automated fashion. This flexibility and automation is what makes AWS CloudFormation and Terraform two popular formats for deploying workloads to Amazon ECS from continuous delivery pipelines.

For more information about AWS CloudFormation, see [Creating Amazon ECS resources with AWS CloudFormation \(p. 395\)](#).

Use AWS CloudFormation or Terraform if you want to automate infrastructure deployments and applications on Amazon ECS and explicitly define and manage all of the AWS resources in your environment.

## AWS Copilot CLI

The AWS Copilot CLI (command line interface) is a comprehensive tool that enables customers to deploy and operate applications packaged in containers and environments on Amazon ECS directly from their source code. When using AWS Copilot you can perform these operations without understanding AWS and Amazon ECS elements such as Application Load Balancers, public subnets, tasks, services, and clusters. AWS Copilot creates AWS resources on your behalf from opinionated service patterns, such as a load balanced web service or backend service, providing an immediate production environment for containerized applications. You can deploy through an AWS CodePipeline pipeline across multiple environments, accounts, or Regions, all of which can be managed within the CLI. By using AWS Copilot you can also perform operator tasks, such as viewing logs and the health of your service. AWS Copilot is an all-in-one tool that helps you more easily manage your cloud resources so that you can focus on developing and managing your applications.

For more information, see [Using the AWS Copilot command line interface \(p. 35\)](#).

Use the AWS Copilot complete end-to-end developer workflow to create, release, and operate container applications that comply with AWS best practices for infrastructure.

## AWS CDK

The AWS Cloud Development Kit (CDK) is an open source software development framework that enables you to model and provision your cloud application resources using familiar programming languages.

AWS CDK provisions your resources in a safe, repeatable manner through AWS CloudFormation. Using the CDK, customers can generate their environment with fewer lines of code using the same language they used to build their application. Amazon ECS provides a module in the CDK that is named `ecs-patterns`, which creates common architectures. An available pattern is `ApplicationLoadBalancedFargateService()`. This pattern creates a cluster, task definition, and additional resources to run a load balanced Amazon ECS service on AWS Fargate.

For more information, see [Getting started with Amazon ECS using the AWS CDK \(p. 16\)](#).

Use the AWS CDK if you want to define infrastructure or architecture as code in your preferred programming language. For example, you can use the same language that you use to write your applications.

## AWS App2Container

Sometimes enterprise customers might already have applications that are hosted on premises or on EC2 instances or both. They are interested in the portability and tooling ecosystem of containers specifically on Amazon ECS, and need to containerize first. AWS App2Container enables you to do just that. App2Container (A2C) is a command line tool for modernizing .NET and Java applications into containerized applications. A2C analyzes and builds an inventory of all applications running in virtual machines, on premises or in the cloud. After you select the application you want to containerize, A2C packages the application artifact and identified dependencies into container images. It then configures the network ports and generates the Amazon ECS task. Last, it creates a CloudFormation template that you can deploy or modify if needed.

For more information, see [Getting started with AWS App2Container](#).

Use App2Container if you have applications that are hosted on premises or on Amazon EC2 instances or both.

## Amazon ECS CLI

The Amazon ECS CLI enables you to run your applications on Amazon ECS and AWS Fargate using the Docker Compose file format. You can quickly provision resources, push and pull images using [Amazon ECR](#), and monitor running applications on Amazon ECS or AWS Fargate. You can also test containers running locally along with containers in the cloud within the CLI.

For more information, see [Using the Amazon ECS command line interface \(p. 41\)](#).

Use the ECS CLI if you have a Compose application and want to deploy it to Amazon ECS, or test local containers with containers running in Amazon ECS in the cloud.

## Docker Desktop integration with Amazon ECS

AWS and Docker have collaborated to make a simplified developer experience that enables you to deploy and manage containers on Amazon ECS directly using Docker tools. You can now build and test your containers locally using Docker Desktop and Docker Compose, and then deploy them to Amazon ECS on Fargate. To get started with the Amazon ECS and Docker integration, download Docker Desktop and optionally sign up for a Docker ID. For more information, see [Docker Desktop](#) and [Docker ID sign up](#).

Beginners to containers often start learning about containers by using Docker tools such as the Docker CLI and Docker Compose. This makes using the Docker Compose CLI plugin for Amazon ECS a

natural next step in running containers on AWS after testing locally. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see [Deploying Docker containers on Amazon ECS](#).

You can take advantage of additional Amazon ECS features, such as service discovery, load balancing and other AWS resources for use with their applications with Docker Desktop.

You can also download the Docker Compose CLI plugin for Amazon ECS directly from GitHub. For more information, see [Docker Compose CLI plugin for Amazon ECS](#) on GitHub.

## AWS SDKs

You can also use AWS SDKs to manage Amazon ECS resources and operations from a variety of programming languages. The SDKs provide modules to help take care of tasks, including tasks in the following list.

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

For more information about the available SDKs, see [Tools for Amazon Web Services](#).

## Summary

With the many options to choose from, you can choose the options that are best suited to you. Consider the following options.

- If you are visually oriented, you can visually create and operate containers using the AWS Management Console.
- If you prefer CLIs, consider using AWS Copilot or the AWS CLI. Alternatively, if you prefer the Docker ecosystem, you can take advantage of the functionality of ECS from within the Docker CLI to deploy to AWS. After these resources are deployed, you can continue managing them through the CLI or visually through the Console.
- If you are a developer, you can use the AWS CDK to define your infrastructure in the same language as your application. You can use the CDK and AWS Copilot to export to CloudFormation templates where you can change granular settings, add other AWS resources, and automate deployments through scripting or a CI/CD pipeline such as AWS CodePipeline.

The AWS CLI, SDKs, or ECS API are useful tools for automating actions on ECS resources, making them ideal for deployment. To deploy applications using AWS CloudFormation you can use a variety of programming languages or a simple text file to model and provision all the resources needed for your applications. You can then deploy your application across multiple Regions and accounts in an automated and secure manner. For example, you can define your ECS cluster, services, task definitions, or capacity providers, as code in a file and deploy through the AWS CLI CloudFormation commands.

To perform operations tasks, you can view and manage resources programmatically using the AWS CLI, SDK, or ECS API. Commands like `describe-tasks` or `list-services` display the latest metadata or a list of all resources. Similar to deployments, customers can write an automation that includes commands such as `update-service` to provide corrective action upon the detection of a resource that has stopped unexpectedly. You can also operate your services using AWS Copilot. Commands like `copilot svc`

`logs` or `copilot app show` provide details about each of your microservices, or about your application as a whole.

Customers can use any of the available tooling mentioned in this document and use them in variety of combinations. ECS tooling offers various paths to graduate from certain tools to use others that fit your changing needs. For example, you can opt for more granular control over resources or more automation as needed. ECS also offers a large range of tools for a wide range of needs and levels of expertise.

## Using the AWS Copilot command line interface

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

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

### Note

The source code for the AWS Copilot CLI is available on [GitHub](#). The latest CLI documentation is available on the AWS Copilot [website](#). We recommend that you submit issues and pull requests for changes that you would like to have included. However, Amazon Web Services doesn't currently support running modified copies of AWS Copilot code. Report issues with AWS Copilot by connecting with us on [Gitter](#) or [GitHub](#) where you can open issues, provide feedback, and report bugs.

## Installing the AWS Copilot CLI

The AWS Copilot CLI can be installed on Linux or macOS systems either by using Homebrew or by manually downloading the binary. Use the following steps with your preferred installation method.

### Installing the AWS Copilot CLI using Homebrew

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

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

### Manually installing the AWS Copilot CLI

As an alternative to Homebrew, you can manually install the AWS Copilot CLI on your macOS or Linux system. Use the following command for your operating system to download the binary, apply execute permissions to it, and then verify it works by listing the help menu.

macOS

For macOS:

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

```
&& copilot --help
```

#### Linux

For Linux x86 (64-bit) systems:

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

For Linux ARM systems:

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

#### Windows

Using Powershell, run the following command:

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

## (Optional) Verify the AWS Copilot CLI using PGP signatures

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

1. Download and install GnuPG. For more information, see the [GnuPG website](#).

#### macOS

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

```
brew install gnupg
```

#### Linux

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

#### Windows

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

For more information, see [GnuPG Download](#).

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

#### macOS

```
echo $PATH
```

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

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

#### Linux

```
echo $PATH
```

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

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

#### Windows

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

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

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

### 3. Create a local plain text file.

#### macOS

On the terminal, enter:

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

Open the file with TextEdit.

#### Linux

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

#### Windows

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

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

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

mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2f174lmavr4Vg
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWfDKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
N+6pPHBJrKIL/1v/ETU4FXpYw2zvhwNahxeNRnoYj3uycHkeliCw4kj0+skizBgO
2K7oVX8Oc3j5+Zilhl/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNiCO
lFTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn
5lBPgfPgRj5hq+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq
lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQq0osYXaQbvlNSW0oc1y/5Zb
zizHTJIhLtUyls9WisP2s0emeHZicVMfW61EgPrJAiupgc7kyZvFt4YwfwARAQAB
tCRBbWVf6b24gRUNTIDx1Y3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhwEEAECAAYF
```

```
AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHnLVpMv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPLt5bVl1+rKqOlw+OgZ4Q/ue/YbWtOAt4qY0OcEoOHgcnaX
lsB827QIfZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmazA
McWB4HUMNrhD0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHYqkVnx4d9UzXF
40oISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
SRERXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLZkNSYqqkpgtwv7seoD2P4n1giRvDAOEfMZpVkuR+C252IaH1HZFEz+TvBVQM
Y8OWWxmIJW+J6evjo3N1eO19UHv71jvoF8zljB14bsL2c+QTJmOv7nRqzDQGCWyp
Id/v2dUVVtK1j9omuLBBwNjzQCb+72LcIzJhYmaP1HC4LcKQG+/f4lexuItenatK
lEJQHytYvXcBlh6Yn/wzNg2NWOw3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPKmyGopsmN/reMuilPdINb249nA0dzoN+nj+ttFOYCIaLaFyjs
Z0r1QA0JAjkEEWEACAMFAlq1SasCGwMHCwkIBWMAQYVCAIJCsEFgIDAQIEaQIX
gAAKCRc86dmkLVF4T9iFEACEnkmlNXsWUx34R3c0vamHrPxfvfyI1fLEUen8D1h
uX9xy6jCEROHWEp0rjGK4QDPgm93sWJ+s1UAKg214QRVzfT0y9/DdR+twApA0fzy
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTzUgVXBL046JhG
eM17+crGUYqeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KftgAsc9rk+
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMrvhVx1
0X90J7cC1KOyhLEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlz3+6xBIm/qe
bFyLUnf4WoiuOplAaJhK9pRY+XENGNxdTn4D26Kd0F+PLkm3Tr3Hy3b1Ok34F1Gr
KVHUq1TZD7cvMnnKEELTUcKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCPvBFhYAlt5Un5zwwkwwQR3/n2kwaODzonJcehdw/C/cGos5D0aIU7I
K222aTD3+pa7Mx3IME2hqmYqRt9X42yF1PIEVrneBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+ekr5TptVszS9few2Gp15bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAxNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8GMTMucIINTH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+JiHwZd8Mx2K+LVVVu/aWkNrfMuNwyDUCiSi4D5QHa8T+F8fgN40TpwYjirzel
5yoICMr9hVcbzDNv/ozKCxj+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXRYtVvYInEhLVrJ0aV6zHFOigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxxAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
t/ModjXr1fJf4pThEi5u6WsuRdFwJY2azEv3vevodTi4HoJREH6dFA6y8c+UDg1
2iHiOKIPqQlbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QuKE+I
R6JA0frUNt2jhIGG/F8RceXzohaac/Cx7LUCUFwC0n7z32C9/Dtj7I1PMOacdZaa
bjJzRKO/ZDv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXP00EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkChWQYAQIAQCUCWrVJqWibDAAKCRc86dmkLVF4T+Zdd/9x/8APzgnJf3o3STrF
jvnVlycyhWYGaeBJiu7wjsNwWzMF0v15tLjB7AqeVxZn+WkDD/mIOQ45OZvnYZuy
X7DR0JszzaH9wrYTxZLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt
Rwe/uwdibIOcagEzyX+2D3kTOlHO5XThbXaNF8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZnl3LSmZyE0EQehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmclioYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+grH3BAUloqF+qK+IUMxTip9KTPNVYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TX1KQ8DBWDhBPVPrRuLIaenTtZEOsPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkfH6+6rbiluJxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUuW
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcW+jsY9IhbEzqN5yQYGi4pVmDkY5vu
lXbJnbqPKPRXGM9BecV9AMbPgbDq/5LnhJJXg+G8YQOqp4lR/hC1TEFdIp5wM8AK
CWSENYt2o1rjgmXiZOMF8A5oBLkCDQRatUuSARAAR77kj7j2QR2SzeOSLFBvV7oS
mFeSNnz9xZssqrs6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+FxqL8hmo9hS4c
rR6tmrP0mOmptr+XLlsKcaP7ogIXsyZnrEAesvW8PnfayoiPCdc3cMCR/1TnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLkvax17PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4b1D32WqvvyESo7vW6WBh7oqv3Zhm0yQrr8a6mDBqpKlvWwNI
3kpJR974tg5o5LfDu1BeeyHWPsgm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTssog+300cGYHV7pWYPhazKHMPm201xKCjH1RfzRULZGKjD+
yMLT1I3AXFmLmZJXikaOlVE3/wgMqCXsbycbLjLD/bXIuFWo3rzoeezeXjgi/DJx
jKBAYBTY05nMcTh109oafD9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuS0sc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfHcVsvbcbcb2Wx+L
IKvmb7EB4K3fmjFFE67yolmiw2qRCUBfygtH3eL5XZU28MiCpue8Y8GKJoBAUyvff
KeM1rO8Jm3iRac5a/D0AEQEAAYkEPGQYAQIAQCUCWrVLkgIbAgIpcRCRc86dmkLVF4
T8FdiaQZaQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bX0ldQR4tKhHq
POLRqy6z1BY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwMOclTb9bBxnBc/4PFvHABZW3QzusaZniNgkuxt6BTflos
E4inq71kjmGK+TlZQ6mUMUG228NUQC+a84EPqYyAeY1sgvgB7hJBhYLOQAxhcW
6m2ORd8iEc8HyZJ3yCOcsKip/nRWAbf0OvfHfRBP0+m0ZwnJM8cPRfjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRlkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bbHlBQyEUB9gKCMUfaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphU17T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLRWvmzQFWzG7AoE55fkf8nAEFsalrCdtanUBHXRxA00QxG
```

```
AHMOdJQQvBsmqMvuAdjkdWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUddewLzGIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vSjKpCNg3EIJSGqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/og5OUif
wcEN1rS9IJXBWly8Me1N9qr5KcKQLmfdBNEYyceBhyV10MDyHOKC+7PoFmTkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yom
aaJu279ioVTrwpECse0XkiRyKToTjwOb73CGkBBZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivhOwWzjpfQKhwcU91ABXi2UvVm14v0AfeI7oiJPSU1zM4fEny4oiIBX1R
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhQUIMii+mWra23EwjChaxpvjjcUH
5ilLc5Zq781aCYRyqYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAx7kPMD3LoF4K1dD
Q8ppQriUvxVo+4nPv6rPtY/PyqCLWDjkguHpJsEFsMkwaJrAz0QNSAU5CJ0G2Zu4
yxvYlumHCEl7nbFrm0vIia75Sa8KnywTdsyZsu3XcOcf3g+glxWtpjJqy2bYXlqz
9uDuOwtArWHOis6bq819RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZY1Nr3lydh+dFHIEkH53HzQe6188HEic
+0jVnLkCDQRa55wJARAAYLya2Lx6gyoWoJN1a6740q3o8e9d4KggQ0fGMTcflmeq
ivuzgN+3DZHN+9ty2KxXMTn0mhHBERZdbNjYjMNT1gAgrhPNB4HtXBxum2wS57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgrJS
Z8t4GpwtDL9gNqyFXaWQmDmkAsCygQMvhAlmu9xOIzQG5CxCsNzFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSVOUW1JojsdCQA
L+RZJcoXq71fOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AyyQ2BbEYU+JAXhUH8rYYui2dHKJigjNvJscuUWb
+QEgJIRleJRhrO+/CHgMs4fZakWF1VFhKBkcKmeJLn1f7EJJUUW84ZhKXjo/AUPX
1CHsNjZiRceUJCJYoxlcwsoq6jTE50GiNzcIxTn9xUC0UMKFeggNAFys1K+TDm3
Bzo8H5ucjCUEmUm9lhkGwqTZgOlRX5eqPX+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6sYKqroYwBGgZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9z1W8qsA
EQEAAyKEPqQYAQgACQUCWuecCQIbAgIpCRC86dmkLVF4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykd/4tOLRHfHXuKucxgGaubUcVtsFrwBKmalcYjqaPms8u
6S10wfgRI32G/GhOrp0Ts/MokbObq6VLTh8N5Yc/53ME18zQFw9Y5AmRow4PZXER
uj5s57p4oR7xHMihMjCCBnlbvrR+34YPfgzTcgLiOEFHYT8UTxwnGmXOVNkMM7md
xD3CV5q6Vate8WKBo/220II3fcQlc9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCMw/NN2gju2G3Lu/TLFUWIT4v/50PK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUHQWeSNHI+fpe5oX6FALPT9JLDce8OZz1i
cZz0MELP37mOQun0AlmHm/hVzf0f311PtbczqWae51tJvgUR/nZFo6Ta305Ezhs
3V1EJNQ1Ijf/6DH87SxvAoRIARCuZd0qxBCDK0avpFzUtbJd241RA3WJpkEimQKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyz0WTDVmt
g0U1CO1ezokMSqk7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtfsCDvALCtqL0jduSkUo+RXcBitmXhA+ShW0pbs2Rtx/ixua
KohVD/OR4QxiSWMICNtm9mw9ydl1lyjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvyZByaWoIe3QxjX84V6M1I2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFfYGBXuAX7+HgPPSFtrHQONCALxxzlbNpS+zzt9r0MiLgcLySpWxSdmoYGZ6nQP
RO5Nm/ZVS+u2imPCRzNUZEMA+dLE6kHxOrS0dPiuJ407NtPeYDKkoQtNagspsDvh
cK7CSqAiKmq06UBTxqlTSRkm62eOCTcs3p3OeHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+45lcCfmcVt94TFNL5HwEUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNJ3bCLuc/jq7ysGq69xiKmTlUeXfm+aojcro5i
zyShIRJZ0GZfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNxl3MdFQE5JJ86rn9
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
```

You may close the text editor.

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



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

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

macOS

For macOS systems, run the following command.

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

Linux

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

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

For Linux ARM systems, run the following command.

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

Windows

Using Powershell, run the following command.

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

7. Verify the signature with the following command.

- For macOS and Linux systems:

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

Expected output:

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

### Important

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

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

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

## Next steps

After installation, learn how to deploy an Amazon ECS application using AWS Copilot. For more information, see [Getting started with Amazon ECS using AWS Copilot \(p. 11\)](#).

# Using the Amazon ECS command line interface

Amazon ECS has released AWS Copilot, a command line interface (CLI) tool that simplifies building, releasing, and operating production-ready containerized applications on Amazon ECS from a local development environment. For more information, see [Using the AWS Copilot command line interface \(p. 35\)](#).

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.

Learn how to use high-level, application-first commands to model, create, release and manage containerized applications from a local development environment at [Getting started with Amazon ECS using AWS Copilot \(p. 11\)](#).

### Topics

- [Installing the Amazon ECS CLI \(p. 41\)](#)
- [Configuring the Amazon ECS CLI \(p. 47\)](#)
- [Migrating Configuration Files \(p. 49\)](#)
- [Tutorial: Creating a cluster with a Fargate task using the Amazon ECS CLI \(p. 50\)](#)
- [Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI \(p. 54\)](#)

## Installing the Amazon ECS CLI

Amazon ECS has released AWS Copilot, a command line interface (CLI) tool that simplifies building, releasing, and operating production-ready containerized applications on Amazon ECS from a

local development environment. For more information, see [Using the AWS Copilot command line interface \(p. 35\)](#).

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.

macOS

```
sudo curl -Lo /usr/local/bin/ecs-cli https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-amd64-latest
```

Linux

```
sudo curl -Lo /usr/local/bin/ecs-cli https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-linux-amd64-latest
```

Windows

Open Windows PowerShell and enter the following commands.

### Note

If you encounter permission issues, ensure that you have administrator access on Windows and you are running PowerShell as an administrator.

```
New-Item -Path 'C:\Program Files\Amazon\ECSCLI' -ItemType Directory  
Invoke-WebRequest -OutFile 'C:\Program Files\Amazon\ECSCLI\ecs-cli.exe' https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-windows-amd64-latest.exe
```

## Step 2: Verify the Amazon ECS CLI using PGP signatures

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

1. Download and install GnuPG. For more information, see the [GnuPG website](#).

macOS

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

```
brew install gnupg
```

Linux

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

Windows

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

For more information, see [GnuPG Download](#).

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

macOS

```
echo $PATH
```

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

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

Linux

```
echo $PATH
```

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

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

Windows

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

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

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

3. Create a local plain text file.

macOS

On the terminal, enter:

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

Open the file with TextEdit.

Linux

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

Windows

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

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

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

mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2f174lmavr4Vg
7K/KH8VH1q2uRw32/B94XLEgRbGTMdWfdKuxoPCttBQaMj3LGn6Pe+6xVVRkChQu
```

BoQAhjBQ+bEmOkNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx  
N+6pHBjKIL/1v/ETU4FXpYw2zvhwNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO  
2K7oVX8Oc3j5+Zilhl/qDLXmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqxXE6NNiCO  
lFTrT7QwD5fMNd3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFxn  
5lBPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9Nl7FJuSsfqbdkvrLRSQRbcBG9qxX3  
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbq  
lTnWWCz4xrIWtuEBA2qE+MlDheVd78a3gIsEaSTfQqOosYXaQbvlNswOoc1y/5Zb  
zizHTJiHlUtUyls9WisP2s0emeHZicVMfW6lEgPrJAiupgc7kyZvFt4YwfwARAQAB  
tCRBbWf6b24gRUNTIDxly3Mtc2VjdXJpdHlAYW1hem9uLmNvbT6JAhWEEAECAAYF  
AlrjL0YACGkQHivRXs0TaQrg1g/+JppwPqHnlVPMv7lessB8I5UqZeD6p6uVpHd7  
Bs3pcPp8BV7BdRbs3sPLt5bVl+rkqOlw+0gZ4Q/ue/YbWtOAt4qY0OcEo0HgcnAX  
lsB827QrfZIVtGWMhuh94xzm/SJkvnngml6KB3YJNnWP61A9qJ37/VbVVLzvcmaZA  
McWB4HUMNRhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJS9kC7OUAHyQkVnx4d9UZXF  
40oISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJX2CSqt7tWJ8gk1n3H3Y  
SRERXJRnv7DsDDBwFgT6r5Q2HW1TBUvaoZy5hF6maD09nHcNnbVjqAdZeT8Tr/Qu  
bBCLZkNSYqqkpgtwv7seoD2P4n1giRvDAOEFmZpVkuR+C252IaH1HZFEZ+TvBVQM  
Y8OWWxmLJW+Jevjo3N1eO19UHv71jvoF8zljbl4bsL2c+QTJmOv7nRqzDQgCWyp  
Id/v2dUVVtk1j9omuLBBWnJzQCB+72LcIzJhYmaP1HC4LcKQG+/f4lexuItenatK  
lEJQhYtyVXcBlh6Yn/wzNg2NWOb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz  
N2HqkTSQh77Z8KPKmyGopsMn/reMUILPdInb249nA0dzoN+nj+ttFOYCIaLaFyjs  
Z0r1QA0AJakeEwECACMFAlq1SasCGwMHCwkIBWMAQYVCAIJCsEFgIDAQIeAQIX  
gAAKCR86dmkLVF4T9iFEACENkmlDNXSWUx34R3c0vamHrPxfvfy1f1fLUEu8D1h  
ux9xy6jCEROHWEp0rjGK4QDPgm93sWJ+s1UAKg214QRVzft0y9/DdR+twApA0fzy  
uavIthGd6+03jAAo6udYDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZUgVXBL046JhG  
eM17+crGUYQeetkiOQemLbsbXQ40Bd9V7zf7XJraFd8VrwNUwNb+9KftgAsc9rk+  
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMrvhVx1  
OX90J7cc1KOyhlEQQ1aFTgmQjMxexVTwIBm8LvysFK6YXm41KjOrlz3+6xBIIm/qe  
bFyLUnf4WoiuOplAaJhK9pRY+XENGNxdN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr  
KVHUq1T2D7cvMnnNKEELTUCkX+1mV3an16nmAg/mylJSUt6BNK2rJpY1s/kkSGSE  
XQ4zuF2IGCpVBFhYAlT5Un5zwwqkwwQR3/n2kwaDzonJcehdw/C/cGos5D0aIU7I  
K2X2aTD3+pa7Mx3IME2hqmYqRt9X42yF1PIEVrNEBRJ3HDezAgJrNh0GQWRQkHix  
gzI/cATR+ekr5TptVsz2GpI5bCgKBKbisZIssT89aw7mAKWutOGcm4Gm9/yK6  
1bkCDQRatUmrARAAXNPvVwreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe  
8uftYZqdC82KeSKhpHq7c8gMTMucIIntH25x9BCc73E33EjCL9Lqov1TL7+QkgHe  
T+JIhZwdD8Mx2K+LVVVu/aWkNrfMuNwyDUCiSI4D5QHa8T+F8fgN40TpwYjirzel  
5yoICMr9hVcbzDNv/ozKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+  
ps1qXRYVvYInEhLVrJ0aV6zHfoigE/Bils6/g7ru1Q6CEHqEw++APS5CcE8VzJu  
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxxAg7rOvyRN9cAXfeSMf77I+XTifigNna8x  
t/ModjXrlfjF4pThEi5u6WsuRdFwjY2azEv3vevodTi4HoJReH6dF8a6y8c+UDgl  
2iHiOKIPqQLbHEfQmHcDd2fix+AaJKMnPGNku9qCFEMbgSRJpXz6BfwnY1QKE+I  
R6JA0frUNT2jhiGG/F8RceXzohaaC/Cx7LUCUFwc0n7z32C9/Dtj7I1PMOacdZzz  
bjzRKO/Zdv+UN/c9dwAkllzAyPMwGBkUaY68EBstnIliw34awm6IiHhxioVPKSp  
VJfyiXPO0EXqujtHLAeChfjcns3I12YshTldv2PafG53fp33ZdzeUgsBo+EAEQEA  
AYkCHwQYAQIACUCWrvJqWbIDAACRC86dmkLVF4T+ZdD/9x/8APzgNjF3o3STrF  
jvnV1ycyhWYGAEbJiu7wjsNWwzMF0v15tLjB7AqeVxZn+WKDD/mIQ450ZvnYZuy  
X7DR0KsZsaH9wrYTxZLVruAu+6tUL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7BlHbvX/gYt  
Rwe/uwdibIOCagEzyX+2D3kTolHO5XThbXanF8AN8zha91Jt2Q2UR2X5T6JcwtMz  
FBvZnl3LSmZyE0E9ehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrqZ0smKNAQng54  
F365W3g8AfY48s8XQwzmcliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S  
XKzr+gRh3BAUloqF+qK+IUMxTip9KTPNVYDpic66yBiT6gFDji5Ca9pGpJXrC3xe  
TxIKQ8DBWDhBPVPrRuLiaenTtZEOPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S  
G5r9ysjkfH6+6rbiliujxMgROSqtqr+RyB+V9A5/OgtNZc8llK6u4UoOCde8jUuW  
vqWKvjJB/Kz3u4zaeNu2ZyyHaOqOuH+TETcw+jsY9IhbEzqN5yQYGi4pVmDkY5vu  
lXbJnbqPKPRXGM9BecV9AMbPgbDq/5LnHJJXg+G8YQOgp41lR/hc1TEFDip5wM8AK  
CWSENYt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAR77kj7j2QR2SZeOSlFBvV7oS  
mFeSNnz9xZssqrsmbTwSHM6YLDwc7Sdf2esDdyZONETwqrVCg+FxgL8hmo9hS4c  
rR6tmrPOMomptr+xlLsKcaP7ogIXsyZnrEAesvW8PnfayoiPCdc3CMCR/1tNHFGA  
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLKvaxl7PNelaHGGYQY/xo+m  
V0bndxf9IY+4oFJ4b1D32WqvyxESo7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI  
3kpJR974tg5o5LfDu1BeeyHWPsgm4U/G4JB+JIG1ADy+RmoWEt4BqTCZ/knnoGvw  
D5sTCxbKdmuOmHgyTssog+300cGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+  
yMLT1I3AXfLmZJXikaOlve3/wgMqCXsbycbLjLD/bXIuFw03rzoeeXjgi/DJx  
jKBAYBTY05nMctH109oafD9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI  
rwuS0sc6XzCzdImj0Wc6axS/HeUKRXWdXJwno5awTwXKRJMXGfhCvSvbcbc2Wx+L  
IKvmb7EB4K3fmjFFE67yolmiw2qRCUBfygth3eL5XZU28MiCpue8Y8GKJoBAUyvf  
KeM1rO8Jm3iRac5a/D0AEQEAAyKEPqYAQIACUCWrvLkgIbAgIpCRC86dmkLVF4

```
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bX01dQR4tKhhQ
POLRqy6z1BY9ILCLowNdGZdqorogUiUymgn3VhEhVtxTOoHCN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOc1Tb9bBxnBc/4PFvHABZW3QZusaZniNgkuxt6BTfloS
Of4inq71kjmGK+TlzQ6mUMQUG228NUQC+a84EPqYyAeY1sgvgB7hJBhYLOQAxhcW
6m20Rd8IEc6HyZJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRikk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJOPGn5SB1HyjCZNx6bbhI1bQyEUB9gKCMUfaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxZophU17T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVXh0mdWYLRWwmzQFWzG7AoE55fkf8nAEPsa1rCdtaNUBHRXA0OQxG
AHMODJQqVbSmqMvuAdjkDWpFu5yOMy5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuce7vsJkPCNg3EIJSGqMOPFjJuLWtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21SsIXk/LqEpLMR/Og5OUif
wcEN1rS9IJXBwIy8Me1N9qr5KcKQLmfdBNEyyceBhyV10MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fclqHV8pwTTo3Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yOM
aaOu279ioVTrwpECse0XkiRyKTOtjWOb73CGkBZzpJyqux/rmCV/fp4ALdSW8zbz
FJVORaiVhoWwzjpfQKhwcU9lABXi2UvVm14v0AfeI7oiJPSU1zm4fEny4oiIBX1R
zhfNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhquIMii+mWra23EwjChaxpvjJcUH
5ilLc5Zq781aCYRyqYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAx7kPMD3LoF4KldD
Q8ppQriUvxVo+4nPv6rPTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCEl7nbFrmOvIiA75Sa8KnywTDSyZsu3XcOcf3g+g1xWtpjJqy2bYXlqz
9uDOWtArWHOis6bq819RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYLnr3lydh+dFHIEkH53HzQe6188HEic
+0jVnLkCDQRa55wJARAAYLya2Lx6gyoWoJN1a6740q3o8e9d4KggQOfGMTCf1meq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBERZdbNjYjMNT1gAgrhPNB4HtXBxum2wS57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IICgclAEyIq0Yq5mAdTEgrJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOizQG5CxSnZFk7zcuL60k14Z3
Cmt49k47/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSV0UW1JojsdCQA
L+RZJcoXq71fOFj/eNjeOSstCTDPfTCL+kTheE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUH8rYYui2dHKJgJNvJscuUWb
+QEeqJIR1eJRhrO+/CHgMs4fZakWF1VFhKBkcKmEjLn1f7EJJUW84ZkXKjo/AUPX
1CHsNjziRceUJCJYoxlcwsoq6jTE50GiNzcIXtn9xUC0UMKFeggNAFys1K+TDtm3
Bzo8H5ucjCUEmUm91lhkGwqT2g0lRX5eqPX+JBosAobghqgCa5IPinKRA6MgoFPHK
6sYKqroYwBGgZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9z1W8qsA
EQEAAYkEPgQYAQgACUCWuecCQIbAgIpCRC86dmkLVF4T8FdIAQZAQgABGUCWuec
CQAKCRBQ3szEcQ5hr+ykd/4tOLRHfHXuKUCxgGaubUcVtsFrwBKma1cYjqqaPms8u
6Sk0wfGRI32G/GhOrp0Ts/M0kbObq6VLTh8N5Yc/53ME18zQFw9Y5AmRow4PZXER
ujs5s7p4oR7xHMiHmJCCBn1bvrR+34YPfgzTcgLiOEfHYT8UTxwnGmXovNkMM7md
xD3CV5q6Vate8WKBo/220II3fcQ1c9r/oWX4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW/NN2gju2G3Lu/TlFUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUHQWeSNHI+fpe5oX6FALPT9JLDce8OZz1i
cZZOMELP37mOQun0AlmHm/hVzf0f311PtbzcgWae51tJvgUR/nZFO6Ta3O5Ezhs
3V1EJNQ11jf/6DH87SxvAoRIARCuZd0qxBCDK0avpFzUtbJd241RA3WJpkEiMqKv
RDVZke4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyzOWTDVmt
g0U1CO1ezokMSqk7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtfsCDvALCtqL0jduSkUo+RXcBiTmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNtm9mw9ydI1lyjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDvvvYZByaWoIe3QxjX84V6MLI2IdAT/xImu8gbaCI8tmyfPirLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxzlbNpS+zxt9r0MiLgcLyspWxSdmoYgZ6nQP
RO5Nm/ZVS+u2imPCRzNUZEMA+dLE6kHxOrS0dPiuJ407NtPeYDKkoQtNagspsDvh
cK7CSqAlKMq06UBTxqlTSRkm62eOCtcs3p30eHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+45lcCfmcVt94TFNL5HwEUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNJ3bCLuc/jq7ysGq69xiKmTlUeXfm+aojcro5i
zyShIRJZ0GZfuzDYFDbMV9amA/YQGygLw//zP5ju5SW26dNxl3MdfQE5JJ86rn9
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
```

You may close the text editor.

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

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

6. Download the 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.

macOS

```
curl -Lo ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-amd64-latest.asc
```

Linux

```
curl -Lo ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-linux-amd64-latest.asc
```

Windows

```
Invoke-WebRequest -OutFile ecs-cli.asc https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-windows-amd64-latest.exe.asc
```

7. Verify the signature.

macOS and Linux

```
gpg --verify ecs-cli.asc /usr/local/bin/ecs-cli
```

Windows

```
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.

macOS and Linux

```
sudo chmod +x /usr/local/bin/ecs-cli
```

Windows

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:

```
setx path "%path%;C:\Program Files\Amazon\ECSCLI"
```

Restart PowerShell so the changes go into effect.

### Note

After 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. 47).

### 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

Amazon ECS has released AWS Copilot, a command line interface (CLI) tool that simplifies building, releasing, and operating production-ready containerized applications on Amazon ECS from a local development environment. For more information, see [Using the AWS Copilot command line interface](#) (p. 35).

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 \(p. 50\)](#). 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. Amazon 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 \(p. 41\)](#).
- Install and configure the AWS CLI. For more information, see [AWS Command Line Interface](#).
- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

### Step 1: Create 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 \(p. 354\)](#).

#### 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. 47).

### 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.

#### Note

Make a note of these IDs as you will need them in the following steps.

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. 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.

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
  os_family: Linux
  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`.

```
ecs-cli compose --project-name tutorial service scale 2 --cluster-config tutorial --ecs-  
profile tutorial-profile
```

In this example, the running count of the application is increased to two.

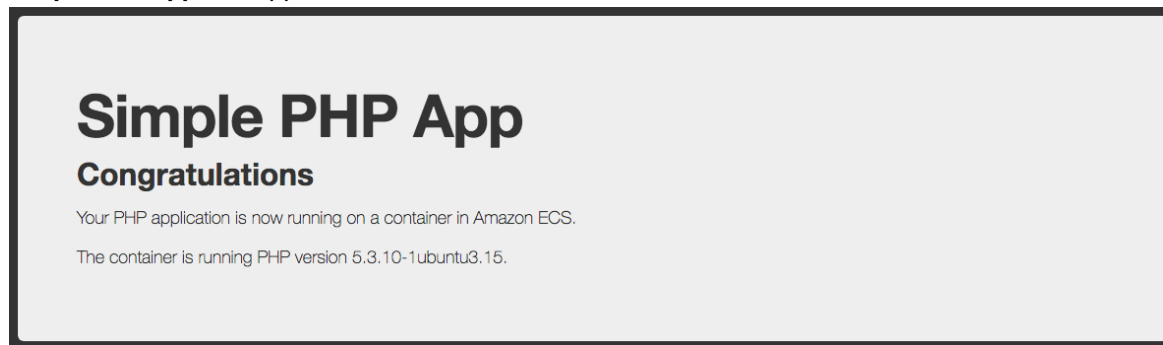
```
ecs-cli compose --project-name tutorial service ps --cluster-config tutorial --ecs-  
profile tutorial-profile
```

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 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 \(p. 41\)](#).

- Create a VPC. For more information, see [the section called “Create a virtual private cloud” \(p. 6\)](#).
- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

## 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 \(p. 41\)](#).

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 \(p. 47\)](#).

### 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
```



Amazon ECS User Guide for AWS Fargate  
Tutorial: Creating an Amazon ECS Service That  
Uses Service Discovery Using the Amazon ECS CLI

```
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-kvhnzhhb5vxplfmls
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
```

# 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, [Task maintenance](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

## Topics

- [Linux platform versions \(p. 58\)](#)
- [Windows platform versions \(p. 62\)](#)

## Linux platform versions

### 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`.

When the **LATEST** platform version is selected, 1.4.0 platform version is used.

- 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 1.4.0 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 \(p. 237\)](#).
- 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.

The following are the available Linux platform versions. For information about platform version deprecation, see [AWS Fargate platform version deprecation \(p. 61\)](#).

## 1.4.0

The following is the changelog for platform version 1.4.0.

- Beginning on November 5, 2020, any new Amazon ECS task launched on Fargate using platform version 1.4.0 will be able to use the following features:

- When using Secrets Manager to store sensitive data, you can inject a specific JSON key or a specific version of a secret as an environment variable or in a log configuration. For more information, see [Specifying sensitive data using Secrets Manager](#) (p. 176).
- Specify environment variables in bulk using the `environmentFiles` container definition parameter. For more information, see [Specifying environment variables](#) (p. 186).
- Tasks run in a VPC and subnet enabled for IPv6 will be assigned both a private IPv4 address and an IPv6 address. For more information, see [Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- The task metadata endpoint version 4 provides additional metadata about your task and container including the task launch type, the Amazon Resource Name (ARN) of the container, and the log driver and log driver options used. When querying the `/stats` endpoint you also receive network rate stats for your containers. For more information, see [Task metadata endpoint version 4](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- Beginning on July 30, 2020, any new Amazon ECS task launched on Fargate using platform version 1.4.0 will be able to route UDP traffic using a Network Load Balancer to their Amazon ECS on Fargate tasks. For more information, see [Service load balancing](#) (p. 252).
- Beginning on May 28, 2020, any new Amazon ECS task launched on Fargate using platform version 1.4.0 will have its ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS owned encryption key. For more information, see [Using data volumes in tasks](#) (p. 127).
- Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see [Amazon EFS volumes](#) (p. 129).
- The ephemeral task storage has been increased to a minimum of 20 GB for each task. For more information, see [Using data volumes in tasks](#) (p. 127).
- 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](#) (p. 136).
- 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. 308).
- 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. 380).
- Added support for the `SYS_PTRACE` Linux parameter in container definitions. For more information, see [Linux parameters](#) (p. 115).
- 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*.
- Based on Amazon Linux 2.

## 1.3.0

The following is the changelog for platform version 1.3.0.

- Beginning on Sept 30, 2019, any new Fargate task that is launched supports the `awsfirelens` log driver. 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 \(p. 144\)](#).
- Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service. For more information, see [Task maintenance](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- Beginning on March 27, 2019, any new Fargate task that is launched can use additional task definition parameters that you use to define a proxy configuration, dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see [Proxy configuration \(p. 119\)](#), [Container dependency \(p. 116\)](#), and [Container timeouts \(p. 117\)](#).
- 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 \(p. 176\)](#).
- 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 \(p. 176\)](#).
- 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 \(p. 109\)](#).
- 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 \(p. 308\)](#).
- Beginning on December 3, 2019, the Fargate Spot capacity provider is supported. For more information, see [AWS Fargate capacity providers \(p. 79\)](#).
- Based on Amazon Linux 2.

## 1.2.0

The following is the changelog for platform version 1.2.0.

### Note

Platform version 1.2.0 is deprecated. We recommend migrating to the latest platform version. For information about platform version deprecation, see [AWS Fargate platform version deprecation \(p. 61\)](#).

- Added support for private registry authentication using AWS Secrets Manager. For more information, see [Private registry authentication for tasks \(p. 174\)](#).

## 1.1.0

The following is the changelog for platform version 1.1.0.

### Note

Platform version 1.1.0 is deprecated. We recommend migrating to the latest platform version. For information about platform version deprecation, see [AWS Fargate platform version deprecation \(p. 61\)](#).

- Added support for the Amazon ECS task metadata endpoint. For more information, see [Amazon ECS task metadata endpoint \(p. 380\)](#).
- Added support for Docker health checks in container definitions. For more information, see [Health check \(p. 104\)](#).
- Added support for Amazon ECS service discovery. For more information, see [Service Discovery \(p. 271\)](#).

## 1.0.0

The following is the changelog for platform version 1.0.0.

### Note

Platform version 1.0.0 is deprecated. We recommend migrating to the latest platform version. For information about platform version deprecation, see [AWS Fargate platform version deprecation](#) (p. 61).

- Based on Amazon Linux 2017.09.
- Initial release.

## Migrating to platform version 1.4.0

The following should be considered when migrating your Amazon ECS on Fargate tasks from platform version 1.0.0, 1.1.0, 1.2.0, or 1.3.0 to platform version 1.4.0. It is considered best practice to confirm your task works properly on platform version 1.4.0 prior to migrating your tasks.

- The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4.0, all Amazon ECS on Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see [Fargate task networking](#) (p. 136).
- 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.

## AWS Fargate platform version deprecation

This page lists platform versions that AWS Fargate has deprecated or have been scheduled for deprecation. These platform versions remain available until the published deprecation date.

A *force update date* is provided for each platform version scheduled for deprecation. On the force update date, any service using the `LATEST` platform version that is pointed to a platform version that is scheduled for deprecation will be updated using the force new deployment option. When the service is updated using the force new deployment option, all tasks running on a platform version scheduled for deprecation are stopped and new tasks are launched using the platform version that the `LATEST` tag points to at that time. Standalone tasks or services with an explicit platform version set are not affected by the force update date.

We recommend updating your services standalone tasks to use the most recent platform version. For more information on migrating to the most recent platform version, see [Migrating to platform version 1.4.0](#) (p. 61).

Once a platform version reaches the *deprecation date*, the platform version will no longer be available for new tasks or services. Any standalone tasks or services which explicitly use a deprecated platform version will continue using that platform version until the tasks are stopped. After the deprecation date, a deprecated platform version will no longer receive any security updates or bug fixes.

Platform version	Force update date	Deprecation date
1.0.0	October 26, 2020	December 14, 2020
1.1.0	October 26, 2020	December 14, 2020
1.2.0	October 26, 2020	December 14, 2020

For information about current platform versions, see [AWS Fargate platform versions \(p. 58\)](#).

## Windows platform versions

### 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 1.0.0, or **LATEST**.

When the **LATEST** platform version is selected the 1.0.0 platform is used.

- 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 \(p. 237\)](#).
- 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.

The following are the available platform versions for Windows containers.

### 1.0.0

The following is the changelog for platform version 1.0.0.

- Initial release for support on the following Microsoft Windows operating systems:
  - Windows Server 2019 Full
  - Windows Server 2019 Core

# New Amazon Elastic Container Service console

Amazon ECS has a new version of the console that is currently in development. the goal of the new console is to simplify deploying container to Amazon ECS by providing smarter defaults, and help panels which guide you through the Amazon ECS configuration process.

You can manage clusters, task definitions, tasks, and services using the new console.

## Topics

- [Cluster management in the new Amazon ECS console \(p. 63\)](#)
- [Task definition management in the new Amazon ECS console \(p. 65\)](#)
- [Task management in the new Amazon ECS console \(p. 70\)](#)
- [Service management in the new Amazon ECS console \(p. 72\)](#)

## Cluster management in the new Amazon ECS console

The following cluster actions are available:

- [Create a task for the Fargate launch type](#)
- [Set the default capacity provider](#)
- [Delete a cluster](#)

## Creating a cluster for the Fargate launch type using the new console

You can create an Amazon ECS cluster using the new AWS Management Console, as described in this topic. Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS \(p. 4\)](#). The new console provides a simple way to create the resources that are needed by an Amazon ECS cluster by creating a AWS CloudFormation stack.

To make the cluster creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

By default, we create an Amazon ECS cluster for Fargate launch type with the following properties:

- Uses Fargate and Fargate Spot capacity providers.
- Launches tasks and services in all the default subnets in the default VPC for your selected Region.
- Does not use Container Insights.
- Has three tags configured for AWS CloudFormation.

You can modify the following default options:

- Change the subnets where tasks and services launch into by default.



- Turn on Container Insights.

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. Container Insights also provides diagnostic information, such as container restart failures, that you use to isolate issues and resolve them quickly. For more information, see [the section called “CloudWatch Container Insights” \(p. 308\)](#).

- Add tags to help you identify your clusters.

### To create a new cluster (New Amazon ECS console)

Before you begin, be sure that you create an IAM user, and then assign the appropriate IAM permissions. For more information, see [the section called “Create an IAM user” \(p. 4\)](#) and [the section called “Cluster examples” \(p. 336\)](#).

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create cluster**.
5. Under **Cluster configuration**, for **Cluster name**, enter a unique name.  
  
The name can contain up to 255 letters (uppercase and lowercase), numbers, and hyphens.
6. (Optional) To change the VPC and subnets where your tasks and services launch, under **Networking**, perform any of the following operations:
  - To remove a subnet, under **Subnets**, choose **X** for each subnet that you want to remove.
  - To change to a VPC other than the **default** VPC, under **VPC**, choose an existing **VPC**, and then under **Subnets**, select each subnet.
7. (Optional) To turn on Container Insights, expand **Monitoring**, and then turn on **Use Container Insights**.
8. (Optional) To manage the cluster tags, expand **Tags**, and then perform one of the following operations:  
  
[Add a tag] Choose **Add tag** and do the following:
  - For **Key**, enter the key name.
  - For **Value**, enter the key value.  
[Remove a tag] Choose **Remove** to the right of the tag's Key and Value.
9. Choose **Create**.

## Setting the cluster default capacity provider using the new console

After the cluster creation completes, you can set the default capacity provider for the cluster. The capacity provider determines the infrastructure that your tasks and services run on.

### To set the default capacity provider for the cluster (New Amazon ECS console)

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose the cluster.

4. On the **Cluster : *name*** page, choose **Edit**.  
You are taken to the classic console.
5. On the **Update cluster** page, choose **Add another provider**.
6. For **Provider 1**, choose the capacity provider, and then choose **Update**.
7. In the navigation pane, choose **Clusters**.  
You are returned to the new console.

## Deleting a cluster using the new console

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

Before you delete a cluster, you must perform the following operations:

- Delete all services in the cluster. For more information, see [the section called “Deleting a service” \(p. 239\)](#).
- Stop all currently running tasks. For more information, see [the section called “Stopping tasks using the new console” \(p. 84\)](#).
- If you created your cluster with the new console, delete the AWS CloudFormation stack that was created for your cluster. The stack is named ***cluster-name*-ECS-Infra**. For example, if the cluster name is "example-cluster-new-console", then the stack name is `example-cluster-new-console-ECS-Infra`. For more information, see [Deleting a stack on the AWS CloudFormation console](#) in the *AWS CloudFormation User Guide*.

If you created a cluster using the classic console and you receive an error when you use the new console, you might need to delete the cluster using the classic console. For more information, see [the section called “Deleting a cluster using the classic console” \(p. 83\)](#).

### To delete a cluster (New Amazon ECS console)

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.
5. In the upper-right of the page, choose **Delete Cluster**.

A message is displayed when you did not delete all the resource associated with the cluster.

6. In the confirmation box, enter **delete *cluster name***.

## Task definition management in the new Amazon ECS console

The following task definition actions are available:

- [Create a task definition](#)
- [Update a task definition](#)
- [Deregister a task definition](#)

## Creating a task definition using the new console

Create your task definitions using the new Amazon ECS console experience. To make the task definition creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

### To create a new task definition (New Amazon ECS console)

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Task definitions**, **Create new task definition**.
3. For **Task definition family**, specify a unique name for the task definition.
4. For each container to define in your task definition, complete the following steps.
  - a. For **Name**, specify a name for the container.
  - b. For **Image URI**, specify the image to use to start a container. Images in the Amazon ECR Public Gallery registry are may be specified using the Amazon ECR Public registry name only. For example, if `public.ecr.aws/ecs/amazon-ecs-agent:latest` is specified, the Amazon Linux container hosted on Amazon ECR Public Gallery is used. For all other repositories, specify the repository using either the `repository-url/image:tag` or `repository-url/image@digest` formats.
  - c. For **Essential container**, if your task definition has two or more containers defined, you may specify whether the container should be considered essential. If a container is marked as *essential*, if that container stops then the task is stopped. Each task definition must contain at least one essential container.
  - d. A port mapping allows the container to access ports on the host to send or receive traffic. Under **Port mappings**, do one of the following:
    - When you use the **awsvpc** network mode, for **Container port** and **Protocol**, specify the port mapping to use for the container.
    - When you use the **bridge** network mode, for **Container port** and **Protocol**, specify the port mapping to use for the container. You select the **bridge** network mode on the next page. After you select it, choose **Previous**, and then for **Host port**, specify the port number on the container instance to reserve for your container.
- Choose **Add more port mappings** to specify additional container port mappings.
- e. Expand the **Environment variables** section to specify environment variables to inject into the container. You can specify environment variables either individually using key-value pairs or in bulk by specifying an environment variable file hosted in an Amazon S3 bucket. For information on how to format an environment variable file, see [Specifying environment variables \(p. 186\)](#).
- f. (Optional) Choose **Add more containers** to add additional containers to the task definition. Choose **Next** once all containers have been defined.
5. For **App environment**, choose **AWS Fargate (serverless)**, **Amazon EC2 instances**, or both. Amazon ECS performs validation using this value to ensure the task definition parameters are valid for the infrastructure type.
6. For **Operating system/Architecture**, choose the operating system and CPU architecture for the task.

To run your task on a 64-bit ARM architecture, select **Linux/ARM64**. For more information, see [the section called "Runtime platform" \(p. 98\)](#).

To run your **AWS Fargate (serverless)** tasks on Windows containers, choose a supported Windows operating system. For more information, see [the section called "Windows containers on AWS Fargate considerations" \(p. 2\)](#).

7. For **Task size**, specify the CPU and memory values to reserve for the task. The CPU value is specified as vCPUs and memory is specified as GB.

For tasks hosted on Fargate, the following table shows the valid CPU and memory combinations.

CPU value	Memory value	
256 (.25 vCPU)	512 MiB, 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	

For tasks hosted on Amazon EC2, supported task CPU 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.

8. Expand the **Container size** section to specify the amount (in GB) of memory to present to the container and the number of CPU units the Amazon ECS container agent will reserve for the container.

If your container attempts to exceed the memory specified, 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.

The total amount of CPU reserved for all containers within a task must be lower than the task-level **CPU** value.

You can multiply the specified value by 1024 to determine the number of CPU units that are available per Amazon EC2 instance type. For example, the value for a t3 nano instance is 2048. For more information, see [Amazon EC2 Instances](#).

9. (Optional) Expand the **Task roles, network mode** section to specify the following:
  - a. For **Task role**, choose the an IAM role to assign to the task. A task IAM role provides permissions for the containers in a task to call AWS APIs.
  - b. For **Network mode**, choose the network mode to use. The default is **awsvpc** mode. For more information, see [Amazon ECS task networking](#).

If you choose **bridge** for the network mode, choose **Previous**, and then under **Port mappings**, for **Host port**, specify the port number on the container instance to reserve for your container.

10. (Optional) The **Storage** section is used to expand the amount of ephemeral storage for tasks hosted on Fargate as well as add a data volume configuration for the task.
  - For **Amount**, to expand the available ephemeral storage beyond the default value of 20 GiB for your Fargate tasks, specify a value up to 200 GiB.
11. (Optional) Choose **Add volume** to add a data volume configuration for the task. For each data volume, complete the following steps.
  - a. For **Volume type**, choose **Bind mount**.

- b. For **Volume name**, specify a name for the data volume. The data volume name is used when creating a container mount point in a later step.
  - c. Expand the **Container mount points** section and choose **Add**.
  - d. For **Container**, choose the container for the mount point.
  - e. For **Source volume**, choose the data volume to mount to the container.
  - f. For **Container path**, specify the path on the container to mount the volume.
  - g. For **Read only**, specify whether to make the volume read only.
  - h. Choose **Add** to add additional mount points until each data volume defined in the task definition has a mount point defined.
12. (Optional) Select the **Use log collection** option to specify a log configuration. For each available log driver, there are log driver options to specify. The default option sends container logs to CloudWatch Logs. The other log driver options are configured using AWS FireLens. For more information, see [Custom log routing \(p. 144\)](#).

The following describes each container log destination in more detail.

- **Amazon CloudWatch** — Configure the task to send container logs to CloudWatch Logs. The default log driver options are provided which creates a CloudWatch log group on your behalf. To specify a different log group name, change the driver option values.
  - **Amazon Kinesis Data Firehose** — Configure the task to send container logs to Kinesis Data Firehose. The default log driver options are provided which sends logs to an Kinesis Data Firehose delivery stream. To specify a different delivery stream name, change the driver option values.
  - **Amazon Kinesis Data Streams** — Configure the task to send container logs to Kinesis Data Streams. The default log driver options are provided which sends logs to an Kinesis Data Streams stream. To specify a different stream name, change the driver option values.
  - **Amazon OpenSearch Service** — Configure the task to send container logs to an OpenSearch Service domain. The log driver options must be provided. For more information, see [Forwarding logs to an Amazon OpenSearch Service domain \(p. 171\)](#).
  - **Amazon S3** — Configure the task to send container logs to an Amazon S3 bucket. The default log driver options are provided but you must specify a valid Amazon S3 bucket name.
13. (Optional) Select the **Use trace collection** option to configure your tasks to route trace data from your application to AWS X-Ray. When this option is selected, Amazon ECS creates an AWS Distro for OpenTelemetry container sidecar which is preconfigured to send the trace data. For more information, see [Collecting application trace data \(p. 310\)](#).

#### Important

When enabling trace collection, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with AWS X-Ray \(p. 310\)](#).

14. (Optional) Select the **Use metric collection** option to collect and send metrics for your tasks to either Amazon CloudWatch or Amazon Managed Service for Prometheus. When this option is selected, Amazon ECS creates an AWS Distro for OpenTelemetry container sidecar which is preconfigured to send the application metrics. For more information, see [Collecting application metrics \(p. 312\)](#).
- a. When **Amazon CloudWatch** is selected, your custom application metrics are routed to CloudWatch as custom metrics. For more information, see [Exporting application metrics to Amazon CloudWatch \(p. 312\)](#).

#### Important

When exporting application metrics to Amazon CloudWatch, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon CloudWatch \(p. 313\)](#).

- b. When **Amazon Managed Service for Prometheus (Prometheus libraries instrumentation)** is selected, your task-level CPU, memory, network, and storage metrics and your custom application metrics are routed to Amazon Managed Service for Prometheus. For **Workspace remote write endpoint**, specify the remote write endpoint URL for your Prometheus workspace. For **Scraping target**, specify the host and port the AWS Distro for OpenTelemetry collector can use to scrape for metrics data. For more information, see [Exporting application metrics to Amazon Managed Service for Prometheus \(p. 315\)](#).

**Important**

When exporting application metrics to Amazon Managed Service for Prometheus, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus \(p. 316\)](#).

- c. When **Amazon Managed Service for Prometheus (OpenTelemetry instrumentation)** is selected, your task-level CPU, memory, network, and storage metrics and your custom application metrics are routed to Amazon Managed Service for Prometheus. For **Workspace remote write endpoint**, specify the remote write endpoint URL for your Prometheus workspace. For more information, see [Exporting application metrics to Amazon Managed Service for Prometheus \(p. 315\)](#).

**Important**

When exporting application metrics to Amazon Managed Service for Prometheus, your task definition requires a task IAM role with the required permissions. For more information, see [Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus \(p. 316\)](#).

15. (Optional) Expand the **Tags** section to add tags, as key-value pairs, to the task definition.
16. Choose **Next** to review the task definition.
17. On the **Review and create** page, review each task definition section. Choose **Edit** to make changes. Once the task definition is complete, choose **Create** to register the task definition.

## Updating a task definition using the new console

A *task definition revision* is a copy of the current task definition with the new parameter values replacing the existing ones. All parameters that you do not modify are in the new revision.

To update a task definition, create a task definition revision. If the task definition is used in a service, you must update that service to use the updated task definition.

When you create a revision, you can modify the following container properties and environment properties.

- Container image URI
- Port mappings
- Environment variables
- Task size
- Container size

### To create a task definition revision (New Amazon ECS 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, choose the task, and then choose **Create new revision**.

5. On the **Create new task definition revision** page, make changes. For example, to change the existing container definitions (such as the container image, memory limits, or port mappings), select the container, and then make the changes.
6. Verify the information, and then 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 \(p. 237\)](#).

## Deregistering a task definition revision using the new console

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. Existing services that reference an `INACTIVE` task definition revision can still scale up or down by modifying the service's desired count.

You can't use an `INACTIVE` task definition revision to run new tasks or create new services. You also can't update an existing service to reference an `INACTIVE` task definition revision (even though there may be up to a 10-minute window following deregistration where these restrictions have not yet taken effect).

### Note

At this time, `INACTIVE` task definition revisions remain discoverable in your account indefinitely. However, this behavior is subject to change in the future. Therefore, you should not rely on `INACTIVE` task definition revisions persisting beyond the lifecycle of any associated tasks and services.

### To deregister a new task definition (New Amazon ECS console)

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. From the navigation bar, choose the region that contains your task definition.
3. In the navigation pane, choose **Task definitions**.
4. On the **task definitions** page, choose the task definition family that contains one or more revisions that you want to deregister.
5. On the **task definition Name** page, choose the revision you want to deregister, such as "example-task:1".
6. In the upper-right of the task definition revision detail page, choose **Deregister**.
7. Verify the information in the **Deregister** window, and then choose **Deregister** to finish.

## Task management in the new Amazon ECS console

The following task actions are available:

- [Run a standalone task](#)

### Run a standalone task using the new console

We recommend that you deploy your application as a standalone task in some situations. For example, suppose that you're developing an application but you're not ready to deploy it with the service



scheduler. If your application is a one-time or periodic batch job, it doesn't make sense to keep running or restart when it finishes.

To deploy your application to run continually or to place it behind a load balancer, create an Amazon ECS service. For more information, see [Amazon ECS services \(p. 212\)](#).

To run a standalone task use one of the following procedures.

If you are creating a Windows service for the Fargate launch type, you must use the classic console.

### To run a standalone task (New Amazon ECS console)

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, select the cluster to run the standalone task in.
4. From the **Tasks** tab, choose **Deploy**.
5. The **Compute configuration** section can be expanded to change the compute option for your service to use. By default, the console selects a compute option for you. Therefore, in most cases, you can proceed to the next step. The following describes the order that the console uses to select a default:
  - If your cluster has a default capacity provider strategy that's defined, the default provider is selected.
  - If your cluster doesn't have a default capacity provider strategy that's defined but you do have the Fargate capacity providers added to the cluster, a custom capacity provider strategy that uses the `FARGATE` capacity provider is selected.
  - If your cluster doesn't have a default capacity provider strategy that's defined but you do have one or more Auto Scaling group capacity providers added to the cluster, the **Use custom (Advanced)** option is selected. You must manually define the strategy.
  - If your cluster doesn't have a default capacity provider strategy that's defined and no capacity providers are added to the cluster, the Fargate launch type is selected.
6. For **Application type**, select **Task**.
7. For **Task definition**, choose the task definition family and revision to use.

**Important**  
The console validates that the selected task definition family and revision is compatible with the defined compute configuration. If you receive a warning, verify that both your task definition compatibility and the compute configuration are selected.
8. For **Desired tasks**, specify the number of tasks to launch.
9. The **Networking** section can be expanded to define the network configuration for the tasks. By default, the console selects the default Amazon VPC along with all subnets and the default security group within the default Amazon VPC. Use the following steps to specify a custom configuration.
  - a. For **VPC**, select the VPC to use.
  - b. For **Subnets**, select one or more subnets in the VPC that the task scheduler is to use when placing your tasks.
  - c. For **Security group**, select an existing security group or create a new one. To use an existing security group, select the security group and move to the next step. To create a new security group, choose **Create a new security group**. You must specify a security group name, description, and add one or more inbound rules for the security group.
  - d. For **Public IP**, choose whether to auto-assign a public IP address to the elastic network interface (ENI) of the task. Tasks that are launched on AWS Fargate can be assigned a public IP address when they're run using a public subnet. This is so that they have a route to the internet. For more information, see [Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.



10. (Optional) The **Tags** section can be expanded to add tags, in the form of key-value pairs, to the service.

## Service management in the new Amazon ECS console

The following service actions are available:

- [Create a service](#)
- [Update a service](#)
- [Delete a service](#)

### Creating a service using the new console

You can create an Amazon ECS service using the new console. To make the service creation process as easy as possible, the console has default selections for many choices which we describe below. There are also help panels available for most of the sections in the console which provide further context.

#### To create a service (New Amazon ECS console)

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster to create the service in.
3. From the **Services** tab, choose **Deploy**.
4. The **Compute configuration** section can be expanded to change the compute option for your service to use. By default, the console will select a compute option for you so in most cases you can go to the next step. The following describes the order that the console uses to select a default:
  - If your cluster has a default capacity provider strategy defined, it will be selected.
  - If your cluster doesn't have a default capacity provider strategy defined but you do have the Fargate capacity providers added to the cluster, a custom capacity provider strategy using the `FARGATE` capacity provider will be selected.
  - If your cluster doesn't have a default capacity provider strategy defined but you do have one or more Auto Scaling group capacity providers added to the cluster, the **Use custom (Advanced)** option is selected and you will need to manually define the strategy.
  - If your cluster doesn't have a default capacity provider strategy defined and no capacity providers added to the cluster, the Fargate launch type is selected.
5. For **Application type**, select **Service**.
6. For **Task definition**, choose the task definition family and revision to use.

**Important**  
The console validates that the selected task definition family and revision is compatible with the defined compute configuration. If you receive a warning, verify both your task definition compatibility and the compute configuration selected.
7. For **Service name**, specify a name for your service.
8. For **Desired tasks**, specify the number of tasks to launch and maintain in the service.
9. The **Deployment options** section can be expanded to change the minimum healthy percent and maximum percent of running tasks allowed during a service deployment. The console has default values for the most common use case selected.

**Note**

Currently, only the **Rolling update** (ECS) deployment type is supported. To use any other deployment type, switch to the classic console.

10. (Optional) The **Load balancing** section can be expanded to configure a load balancer for your service. Use the following steps to configure your service to use an Application Load Balancer.
  - a. For **Load balancer type**, select **Application Load Balancer**.
  - b. Choose **Create a new load balancer** to create a new Application Load Balancer or **Use an existing load balancer** to select an existing Application Load Balancer.
  - c. When creating a new load balancer, for **Load balancer name**, specify a unique name for your load balancer. When using an existing load balancer, for **Load balancer**, select your existing load balancer.
  - d. For **Listener**, specify a port and protocol for the Application Load Balancer to listen for connection requests on. By default, the load balancer will be configured to use port 80 and HTTP.
  - e. For **Target group name**, specify a name and a protocol for the target group that the Application Load Balancer will route requests to. By default, the target group will route requests to the first container defined in your task definition.
  - f. For **Health check path**, specify a path that exists within your container where the Application Load Balancer should periodically send requests to verify the connection health between the Application Load Balancer and the container. By default, a path of / is used which is the root directory.
  - g. For **Health check grace period**, specify the amount of time (in seconds) that the service scheduler should ignore unhealthy Elastic Load Balancing target health checks for.
11. The **Networking** section can be expanded to define the network configuration for the service. Task definitions that use the `awsvpc` network mode or services configured to use a load balancer must have a networking configuration. By default, the console selects the default Amazon VPC along with all subnets and the default security group within the default Amazon VPC. Use the following steps to specify a custom configuration.
  - a. For **VPC**, select the VPC to use.
  - b. For **Subnets**, select one or more subnets in the VPC that the task scheduler should consider when placing your tasks.
  - c. For **Security group**, you can either select an existing security group or create a new one. To use an existing security group, select the security group and move to the next step. To create a new security group, choose **Create a new security group**. You must specify a security group name, description, and then add one or more inbound rules for the security group.
  - d. For **Public IP**, choose whether to auto-assign a public IP address to the elastic network interface (ENI) of the task. Tasks that are launched on AWS Fargate can be assigned a public IP address when run using a public subnet so they have a route to the internet. For more information, see [Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
12. (Optional) The **Tags** section can be expanded to add tags, in the form of key-value pairs, to the service.

## Updating a service using the new console

You can update an Amazon ECS service using the new console. When updating a service using the AWS Management Console, the current service configuration is pre-populated. You are able to update the task definition, desired task count, capacity provider strategy, platform version, and deployment configuration; or any combination of these.

### Note

Currently, only services using the **Rolling update** (ECS) deployment type should be updated using the new console. To update a service using any other deployment type, switch to the old console.

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 service (New Amazon ECS console)

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster.
3. On the **Cluster overview** page, check the box next to the service to update and choose **Edit**.
4. For **Task definition**, choose the task definition family and revision to use.

#### Important

The console validates that the selected task definition family and revision is compatible with the defined compute configuration. If you receive a warning, verify both your task definition compatibility and the compute configuration selected.

5. Expand the **Deployment options** section and use the following steps to change the deployment configuration for your service.
  - a. For services on AWS Fargate the platform version can be updated.
  - b. For services using a capacity provider strategy, the capacity provider strategy can be updated.

#### Note

A service using an Auto Scaling group capacity provider can't be updated to use a Fargate capacity provider and vice versa.

- c. Select the **Force new deployment** option to have your service start a new deployment, which will stop all currently running tasks and launch new tasks using the updated configuration.
  - d. For **Min running tasks**, specify the lower limit on the number of tasks in the service that must remain in the **RUNNING** state during a deployment, as a percentage of the desired number of tasks (rounded up to the nearest integer). For more information, see [Deployment configuration](#).
  - e. For **Max running tasks**, specify the upper limit on the number of tasks in the service that are allowed in the **RUNNING** or **PENDING** state during a deployment, as a percentage of the desired number of tasks (rounded down to the nearest integer).
6. Expand the **Tags** section to update the tags associated with the service.
  7. Choose **Update**.

## Deleting a service using the new console

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 a service (New Amazon ECS console)**

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster for the service.
3. On the **Clusters** page, choose the cluster.
4. On the **Cluster : *name*** page, choose the **Services** tab.
5. Select the services, and then choose **Delete**.
6. At the confirmation prompt, enter **delete me** and then choose **Delete**.

# Amazon ECS clusters

An Amazon ECS cluster is a logical grouping of tasks or services. Your tasks and services are run on infrastructure that is registered to a cluster. The infrastructure capacity can be provided by AWS Fargate, which is serverless infrastructure that AWS manages, Amazon EC2 instances that you manage, or an on-premise server or virtual machine (VM) that you manage remotely. In most cases, Amazon ECS capacity providers can be used to manage the infrastructure the tasks in your clusters use. For more information, see [Amazon ECS capacity providers \(p. 78\)](#).

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 \(p. 76\)](#)
- [Creating a cluster using the classic console \(p. 77\)](#)
- [Amazon ECS capacity providers \(p. 78\)](#)
- [Updating cluster settings \(p. 83\)](#)
- [Deleting a cluster using the classic console \(p. 83\)](#)
- [Stopping tasks using the new console \(p. 84\)](#)

## 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 hosted on AWS Fargate, Amazon EC2 instances, or external instances. For more information about launch types, see [Amazon ECS launch types](#) (p. 124).
- 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 capacity providers](#) (p. 78).
- Custom IAM policies may be created to allow or restrict user access to specific clusters. For more information, see the [Cluster examples](#) (p. 336) section in [Identity-based policy examples for Amazon Elastic Container Service](#) (p. 331).
- Clusters with Fargate tasks can be scaled using Service Auto Scaling. For more information, see [Service auto scaling](#) (p. 263).

## Creating a cluster using the classic console

You can create an Amazon ECS cluster using the classic AWS Management Console, as described in this topic. Before you begin, be sure that you've completed the steps in [Set up to use Amazon ECS](#) (p. 4).

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. If your requirements extend beyond what is supported in this wizard, consider using our reference architecture at <https://github.com/aws-labs/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 (AWS Management Console)

Before you begin, be sure that you create an IAM user, and then assign the appropriate IAM permissions. For more information, see [the section called "Create an IAM user"](#) (p. 4) and [the section called "Cluster examples"](#) (p. 336).

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 **Networking only**, then choose **Next Step**.

#### Important

The `FARGATE` and `FARGATE_SPOT` capacity providers will be automatically associated with the cluster. For more information, see [AWS Fargate capacity providers](#) (p. 79).

6. On the **Configure cluster** page, enter a **Cluster name**. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
7. 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.
8. In the **CloudWatch Container Insights** section, choose whether to turn on Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights](#) (p. 308).
9. Choose **Create**.

## Amazon ECS capacity providers

Amazon ECS capacity providers are used to manage the infrastructure the tasks in your clusters use. Each cluster can have one or more capacity providers and an optional default capacity provider strategy. The capacity provider strategy determines how the tasks are spread across the cluster's capacity providers. When you run a standalone task or create a service, you may either use the cluster's default capacity provider strategy or specify a capacity provider strategy that overrides the cluster's default strategy.

### Capacity provider concepts

Capacity providers consist of the following components.

#### Capacity provider

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

For Amazon ECS on AWS Fargate users, there is a `FARGATE` and a `FARGATE_SPOT` capacity provider. The AWS Fargate capacity providers are reserved and don't need to be created nor can they be deleted. After you associate them with your cluster, you may add them to a capacity provider strategy. For more information, see [AWS Fargate capacity providers \(p. 79\)](#).

#### Default capacity provider strategy

A *default capacity provider strategy* is associated with an Amazon ECS cluster. This determines the capacity provider strategy used creating a service or running a standalone task in the cluster when there isn't a custom capacity provider strategy or launch type specified. It is considered best practice to define a default capacity provider strategy for each cluster.

#### Capacity provider strategy

A *capacity provider strategy* is specified when creating a service or running a standalone task when the default capacity provider strategy for a cluster does not meet your needs.

Only capacity providers that are already associated with a cluster and have an `ACTIVE` or `UPDATING` status can be used in a capacity provider strategy. A capacity provider can be associated with a cluster either during cluster creation or by using the `PutClusterCapacityProviders` API after a cluster has been created.

A capacity provider strategy consists of one or more capacity providers. An optional *base* and *weight* value may be specified for finer control of a capacity 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*.

### Capacity provider types

The infrastructure your Amazon ECS workloads are run on determines the type of capacity provider you can use.

For Amazon ECS workloads hosted on Fargate, the following predefined capacity providers are available:

- Fargate
- Fargate Spot

For Amazon ECS workloads hosted on Amazon EC2 instances, you must create and maintain a capacity provider that consists of the following components:

- A name
- An Auto Scaling group
- The settings for managed scaling and managed termination protection.

## Capacity provider considerations

The following should be considered when using capacity providers:

- A capacity provider must be associated with a cluster prior to being specified in a capacity provider strategy.
- When you specify a capacity provider strategy, the number of capacity providers that can be specified is limited to six.
- A service using an Auto Scaling group capacity provider can't be updated to use a Fargate capacity provider and vice versa.
- In a capacity provider strategy, if no `weight` value is specified for a capacity provider in the console then the default value of 1 is used. If using the API or AWS CLI, the default value of 0 is used.
- When multiple capacity providers are specified within a capacity provider strategy, at least one of the capacity providers must have a weight value greater than zero and any capacity providers with a weight of 0 will not be used to place tasks. If you specify multiple capacity providers in a strategy that all have a weight of 0, any `RunTask` or `CreateService` actions using the capacity provider strategy will fail.
- In a capacity provider strategy, only one capacity provider can have a *base* value defined. If no base value is specified, the default value of 0 is used.
- A cluster may contain a mix of both Auto Scaling group capacity providers and Fargate capacity providers, however a capacity provider strategy may only contain one or the other but not both.
- A cluster may contain a mix of services and standalone tasks using both capacity providers and launch types. A service may 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 use managed termination protection, you must also use managed scaling otherwise managed termination protection won't work.
- Using capacity providers is not supported when using Classic Load Balancers for your services.

## AWS Fargate capacity providers

Amazon ECS on AWS Fargate capacity providers allow you to use both Fargate and Fargate Spot capacity with your Amazon ECS tasks. For more information about capacity providers, see [Amazon ECS capacity providers \(p. 78\)](#).

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.

## Fargate capacity provider considerations

The following should be considered when using Fargate capacity providers.



- The Fargate Spot capacity provider is not supported for Windows containers on Fargate.
- The Fargate Spot capacity provider is not supported for Linux tasks with the ARM64 architecture, Fargate Spot only supports Linux tasks with the X86\_64 architecture.
- The Fargate and Fargate Spot capacity providers don't need to be created. They are available to all accounts and only need to be associated with a cluster to be available for use.
- To associate Fargate and Fargate Spot capacity providers to an existing cluster, you must use the Amazon ECS API or AWS CLI. For more information, see [Adding Fargate capacity providers to an existing cluster \(p. 81\)](#).
- The Fargate and Fargate Spot capacity providers are reserved and cannot be deleted. You can disassociate them from a cluster using the `PutClusterCapacityProviders` API.
- When a new cluster is created using the Amazon ECS classic console along with the **Networking only** cluster template, the `FARGATE` and `FARGATE_SPOT` capacity providers are associated with the new cluster automatically.
- Using Fargate Spot requires that your task use platform version 1.3.0 or later (for Linux). For more information, see [AWS Fargate platform versions \(p. 58\)](#).
- 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 \(p. 299\)](#).
- 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. A service with only one task will be interrupted until 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. If you don't specify a `stopTimeout` value, the default value of 30 seconds is used. For more information, see [Container timeouts \(p. 117\)](#).
- The `SIGTERM` signal must be received from within the container to perform any cleanup actions. Failure to process this signal will result in the task receiving a `SIGKILL` signal after the configured `stopTimeout` and may result in data loss or corruption.

The following is a snippet of a task state change event 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 can specify one or more capacity providers to associate with the cluster. The capacity providers are used to define a capacity provider strategy which determine the infrastructure your tasks run 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 using the classic console](#) (p. 77).

### 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
```

### Note

When running standalone tasks using Fargate Spot it is important to note that the task may be interrupted before it is able to complete and exit. It is therefore important that you code your

application to gracefully exit within 2 minutes when it receives a SIGTERM signal and be able to be resumed. For more information, see [Handling Fargate Spot termination notices](#) (p. 80).

## 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 allow 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 turn on or off CloudWatch Container Insights for an existing cluster. To turn on 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 using the classic console](#) (p. 77).

### 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 turn on Container Insights, you must wait 7 days before you can re-create it.

### To update the settings for a cluster (AWS CLI)

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 using the classic console

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

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. This condition is discussed in step 5 of the following procedure.

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 AWS CloudFormation stack](#) in the *AWS CloudFormation User Guide*.

#### To delete a cluster using the classic console

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.
5. In the upper-right of the page, choose **Delete Cluster**. You see a confirmation prompt.
6. In the confirmation box, enter **delete me**.

## Stopping tasks using the new console

If you decide that you no longer need to keep a task running, you can use the new console to stop one or more tasks.

#### To stop tasks using the new 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 the cluster.
4. On the **Cluster : *name*** page, choose the **Tasks** tab.
5. Perform one of the following operations:
  - To stop one or more tasks, select the tasks, and then choose **Stop, Stop selected**.
  - To stop all tasks, choose **Stop, Stop all**.
6. On the **Stop confirmation page**, enter **Stop**, and then choose **Stop**.

# Amazon ECS task definitions

A task definition is required to run Docker containers in Amazon ECS. The following are some of the parameters that you can specify in a task definition:

- The Docker image to use with each container in your task
- How much CPU and memory to use with each task
- The launch type to use, which determines the infrastructure that your tasks are hosted on
- The operating system of the container that the task runs on
- The Docker networking mode to use for the containers in your task
- The logging configuration to use for your tasks
- Whether the task continues to run if the container finishes or fails
- The command that the container runs when it's started
- Any data volumes that are used with the containers in the task
- The IAM role that your tasks use

Your entire application stack doesn't need to be on a single task definition, and in most cases it isn't on a single task definition. Your application can span multiple task definitions. You can do this by combining related containers into their own task definitions, each representing a single component. For more information, see [Application architecture](#) (p. 90).

## Topics

- [Fargate task definition considerations](#) (p. 85)
- [Application architecture](#) (p. 90)
- [Creating a task definition using the classic console](#) (p. 91)
- [Task definition parameters](#) (p. 97)
- [Amazon ECS launch types](#) (p. 124)
- [Working with 64-bit ARM workloads on Amazon ECS](#) (p. 125)
- [Using data volumes in tasks](#) (p. 127)
- [Fargate task networking](#) (p. 136)
- [Using the awslogs log driver](#) (p. 138)
- [Custom log routing](#) (p. 144)
- [Private registry authentication for tasks](#) (p. 174)
- [Specifying sensitive data](#) (p. 176)
- [Specifying environment variables](#) (p. 186)
- [Example task definitions](#) (p. 189)
- [Updating a task definition using the classic console](#) (p. 194)
- [Deregistering a task definition revision](#) (p. 194)

## Fargate task definition considerations

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

The following task definition parameters aren't valid in Fargate tasks:

- `devices`
- `disableNetworking`
- `dnsSearchDomains`
- `dnsServers`
- `dockerSecurityOptions`
- `dockerVolumeConfiguration`
- `extraHosts`
- `host`
- `hostname`
- `links`
- `placementConstraints` — By default, Fargate tasks are spread across multiple Availability Zones.
- `privileged`
- `sharedMemorySize`
- `tmpfs`

### Important

If one of the task definition parameters isn't supported, the subflags for that parameter likely aren't supported either.

The following task definition parameters behave differently for Fargate tasks:

- When using `logConfiguration`, the supported log drivers for Fargate tasks are the `awslogs`, `splunk`, and `awsfirelens` log drivers.  
  
Windows tasks don't support `awsfirelens` log drivers.
- When using `linuxParameters` for `capabilities`, the `drop` parameter can be used, but the `add` parameter isn't supported.
- The `healthCheck` parameter is supported only for Fargate tasks that use the platform version 1.1.0 or later.
- If you use the `portMappings` parameter, specify `containerPort` only. You can either keep `hostPort` blank or set it to the same value as `containerPort`.
- The `operatingSystemFamily` parameter is required for tasks that run on Linux containers and Windows containers.

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

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

## Network mode

Fargate task definitions require that the network mode is set to `awsvpc`. The `awsvpc` network mode provides each task with its own elastic network interface. A network configuration is also required when creating a service or manually running tasks. For more information, see [Fargate Task Networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

## Task CPU and memory

For Fargate task definitions, you're required to specify CPU and memory at the task level. You can also specify CPU and memory at the container level for Fargate tasks. However, doing so isn't required. For most use cases, it's OK to specify these resources at the task level only. The following table 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 (0.5 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

## Logging

Fargate task definitions support the `awslogs`, `splunk` and `awsfirelens` log drivers only for the log configuration. The following code 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 how you can use the `awslogs` log driver in task definitions to send your container logs to CloudWatch Logs, see [Using the awslogs log driver \(p. 138\)](#).

For more information about using the `awsfirelens` log driver in a task definition, see [Custom log routing \(p. 144\)](#).

## Amazon ECS Task execution IAM role

You can specify an optional task execution IAM role with Fargate to allow your Fargate tasks to make API calls to Amazon ECR. The API calls pull container images. They also call CloudWatch to store container application logs. For more information, see [Amazon ECS task execution IAM role \(p. 354\)](#).

## Example task definition

The following is an example task definition that uses the Linux containers on Fargate launch type to set 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",
"runtimePlatform": {
    "operatingSystemFamily": "LINUX"
},
"requiresCompatibilities": [
    "FARGATE"
]
}

```

## Task storage

For Amazon ECS tasks on Fargate, the following storage types are supported:

- Amazon EFS volumes for persistent storage. For more information, see [Amazon EFS volumes \(p. 129\)](#).
- Ephemeral storage for nonpersistent storage.

When provisioned, each Amazon ECS task hosted 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 \(p. 281\)](#).

### Note

The `host` and `sourcePath` parameters are not supported for Fargate tasks.

## Fargate Linux platform versions

### 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 GiB of ephemeral storage. Both the pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage your task has to use, you must subtract the amount of storage your container image uses from the total amount.

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

#### Note

This amount includes both compressed and uncompressed container image artifacts.

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

## Fargate Windows platform versions

### Fargate tasks using platform version 1.0.0 or later

All Amazon ECS on Fargate tasks using platform version 1.0.0 or later receive a minimum of 20 GiB of ephemeral storage. Both the pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage your task has to use, you must subtract the amount of storage your container image uses from the total amount.

The ephemeral storage is encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key.

## Example task definition

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

### To provide nonpersistent empty storage for containers in a Fargate task

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": "amazon/amazon-ecs-sample",  
    "memory": 128,  
    "cpu": 1024,  
    "essential": true,  
    "mountPoints": [  
      {  
        "containerPath": "/scratch",  
        "sourcePath": "/scratch",  
        "readOnly": false  
      }  
    ],  
    "volumesFrom": [  
      "application2"  
    ]  
  },  
  {  
    "name": "application2",  
    "image": "amazon/amazon-ecs-sample",  
    "memory": 128,  
    "cpu": 1024,  
    "essential": true,  
    "mountPoints": [  
      {  
        "containerPath": "/scratch",  
        "sourcePath": "/scratch",  
        "readOnly": false  
      }  
    ],  
    "volumesFrom": [  
      "application1"  
    ]  
  }  
]
```

```
{
  "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"
    }
  ]
}
]
```

## Application architecture

You can follow one of the two following models to run your containers:

- Fargate launch type - This is a serverless pay-as-you-go option. You can run containers without having to manage your infrastructure.
- EC2 launch type - Configure and deploy EC2 instances in your cluster to run your containers.

How you architect your application on Amazon ECS depends on several factors, with the launch type that you're using being a key differentiator. We give the following guidance which should assist in the process.

### Using the Fargate launch type

The Fargate launch type is suitable for the following workloads:

- Large workloads that require low operational overhead
- Small workloads that have occasional burst
- Tiny workloads
- Batch workloads

When architecting your application to run on Amazon ECS using AWS Fargate, you must decide between deploying multiple containers into the same task definition and deploying containers separately in multiple task definitions.

If the following conditions are required, we recommend deploying multiple containers into the same task definition:

- Your containers share a common lifecycle (that is, they're launched and terminated together).

- Your containers must run on the same underlying host (that is, one container references the other on a localhost port).
- You require that your containers share resources.
- Your containers share data volumes.

If these conditions aren't required, we recommend deploying containers separately in multiple task definitions. This is because, by doing so, you can scale, provision, and deprovision them separately.

## Creating a task definition using the classic console

### Important

Amazon ECS has provided a new console experience for creating task definitions. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

Before running Docker containers on Amazon ECS, you must first create a task definition. When you create a task definition, you can use it to define multiple containers and data volumes. For more information about the available parameters for task definitions, see [Task definition parameters \(p. 97\)](#).

### To create a new task definition (classic Amazon ECS console)

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 compatibilities** page, choose **FARGATE** and then **Next step**.
4. (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**.

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 **Operating system family**, choose the container operating system.
7. 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 \(p. 354\)](#).
8. 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	Operating systems supported for Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows

CPU value	Memory value	Operating systems supported for Fargate
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows

9. 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 \(p. 97\)](#).
  - c. Choose **Add** to add your container to the task definition.
10. (Optional) For **Service Integration**, to configure the parameters for App Mesh integration, choose **Enable App Mesh integration** and then do the following:
  - a. For **Mesh name**, choose the existing App Mesh service mesh to use. If you don't see any meshes listed, then you need to create one first. For more information, see [Service meshes](#) in the *AWS App Mesh User Guide*.

**Note**  
This option is not available for Windows containers on Fargate.
  - b. For **App Mesh endpoints**, select one of the following options.
    - **Virtual node** – Enter or select the following information.
      - For **Application container name**, choose the container name to use for the App Mesh integration. This container must already be defined within the task definition.
      - For **Virtual node name**, choose the existing App Mesh virtual node to use. If you don't see any virtual nodes listed, then you need to create one first. For more information, see [Virtual nodes](#) in the *AWS App Mesh User Guide*.
      - For **Virtual node port** – Pre-populated with the listener port set on the virtual node in App Mesh.
    - **Virtual gateway** – Enter or select the following information.
      - For **Virtual gateway name**, choose the existing App Mesh virtual gateway to use. If you don't see any virtual gateways listed, then you need to create one first. For more information, see [Virtual gateways](#) in the *AWS App Mesh User Guide*.
      - For **Virtual gateway port** – Pre-populated with the listener port set on the virtual gateway in App Mesh.
  - c. For **Envoy image**, enter `840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod` for all regions except `me-south-1` and `ap-east-1`. You can replace `us-west-2` with any Region except `me-south-1` and `ap-east-1`. If your application is in one of these regions, then you also need to replace `840364872350` with the appropriate value for your Region. For more information, see [Envoy image](#) in the *AWS App Mesh User Guide*.
  - d. Choose **Apply** and then choose **Confirm**. This will add an Envoy proxy container to the task definition, as well as the settings to support it. If you selected **Virtual node**, it will also auto-populate the App Mesh **Proxy Configuration** settings for the next step. If you selected **Virtual gateway**, then the **Proxy Configuration** is disabled, because it's not used for a virtual gateway.
11. (Optional) If you selected **Virtual node** in **Service Integration**, then for **Proxy Configuration**, verify all of the pre-populated values. For more information about these fields, see the JSON tab in [Update services](#).
12. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:

### Note

This option is not available for Windows containers on Fargate.

- 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 pre-populates with the AWS for Fluent Bit image. For more information, see [Using the AWS for Fluent Bit image \(p. 148\)](#).
  - 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.
13. (Optional) To define data volumes for your task, choose **Add volume**. For more information, see [Using data volumes in tasks \(p. 127\)](#).
    - For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
  14. 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](#).
  15. Choose **Create**.

### To create a new task definition (AWS CLI)

- Use the `register-task-definition` command. For more information, see [register-task-definition](#) in the *AWS Command Line Interface Reference*.

## Task definition template

An empty task definition template is shown as follows. You can use this template to create your task definition, which can then be pasted into the console JSON input area or saved to a file and used with the AWS CLI `--cli-input-json` option. For more information, see [Task definition parameters \(p. 97\)](#).

### Important

You must include the `operatingSystemFamily` parameter with one of the following values:

- `LINUX`
- `WINDOWS_SERVER_2019_FULL`
- `WINDOWS_SERVER_2019_CORE`

```
{
  "family": "",
  "runtimePlatform": {
    "operatingSystemFamily": ""
  },
  "taskRoleArn": "",
  "executionRoleArn": "",
  "networkMode": "awsvpc",
  "platformFamily": "",
  "containerDefinitions": [
    {
      "name": "",
      "image": "",
      "repositoryCredentials": {"credentialsParameter": ""},
      "cpu": 0,
```

```

"memory": 0,
"memoryReservation": 0,
"links": [""],
"portMappings": [
  {
    "containerPort": 0,
    "hostPort": 0,
    "protocol": "tcp"
  }
],
"essential": true,
"entryPoint": [""],
"command": [""],
"environment": [
  {
    "name": "",
    "value": ""
  }
],
"environmentFiles": [
  {
    "value": "",
    "type": "s3"
  }
],
"mountPoints": [
  {
    "sourceVolume": "",
    "containerPath": "",
    "readOnly": true
  }
],
"volumesFrom": [
  {
    "sourceContainer": "",
    "readOnly": true
  }
],
"linuxParameters": {
  "capabilities": {
    "add": [""],
    "drop": [""],
  },
  "devices": [
    {
      "hostPath": "",
      "containerPath": "",
      "permissions": ["read"]
    }
  ],
  "initProcessEnabled": true,
  "sharedMemorySize": 0,
  "tmpfs": [
    {
      "containerPath": "",
      "size": 0,
      "mountOptions": [""],
    }
  ],
  "maxSwap": 0,
  "swappiness": 0
},
"secrets": [
  {
    "name": "",
    "valueFrom": ""
  }
]

```

```

    }
  ],
  "dependsOn": [
    {
      "containerName": "",
      "condition": "HEALTHY"
    }
  ],
  "startTimeout": 0,
  "stopTimeout": 0,
  "hostname": "",
  "user": "",
  "workingDirectory": "",
  "disableNetworking": true,
  "privileged": true,
  "readonlyRootFilesystem": true,
  "dnsServers": [""],
  "dnsSearchDomains": [""],
  "extraHosts": [
    {
      "hostname": "",
      "ipAddress": ""
    }
  ],
  "dockerSecurityOptions": [""],
  "interactive": true,
  "pseudoTerminal": true,
  "dockerLabels": {"KeyName": ""},
  "ulimits": [
    {
      "name": "msgqueue",
      "softLimit": 0,
      "hardLimit": 0
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {"KeyName": ""},
    "secretOptions": [
      {
        "name": "",
        "valueFrom": ""
      }
    ]
  },
  "healthCheck": {
    "command": [""],
    "interval": 0,
    "timeout": 0,
    "retries": 0,
    "startPeriod": 0
  },
  "systemControls": [
    {
      "namespace": "",
      "value": ""
    }
  ],
  "resourceRequirements": [
    {
      "value": "",
      "type": "GPU"
    }
  ],
  "firelensConfiguration": {
    "type": "fluentd",

```



```

        "options": {"KeyName": ""}
    }
},
"volumes": [
    {
        "name": "",
        "host": {"sourcePath": ""},
        "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": ["FARGATE"],
"cpu": "",
"memory": "",
"tags": [
    {
        "key": "",
        "value": ""
    }
],
"ephemeralStorage": {
    "sizeInGiB": 0
},
"pidMode": "task",
"ipcMode": "none",
"proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "",
    "properties": [
        {
            "name": "",
            "value": ""
        }
    ]
},
"inferenceAccelerators": [
    {
        "deviceName": "",
        "deviceType": ""
    }
]
}

```

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

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

## 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. In contrast, task role, network mode, volumes, task placement constraints, and launch type are optional.

You can use these parameters in a JSON file to configure your task definition. For more information, see [the section called “Example task definitions” \(p. 169\)](#).

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's registered into a particular family is given a revision of 1, and any task definitions registered after that are given a sequential revision number.

### Launch types

When you register a task definition, you can specify a launch type that Amazon ECS should validate the task definition against. If the task definition doesn't validate against the compatibilities specified, a client exception is returned. For more information, see [Amazon ECS launch types \(p. 124\)](#).

The following parameter is allowed in a task definition.

requiresCompatibilities

Type: string array

Required: no

Valid Values: EC2 | FARGATE | EXTERNAL

The launch type to validate the task definition against. This enables a check to ensure that all of the parameters that are used in the task definition meet the requirements of the launch type.

### 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. 354).

## Network mode

`networkMode`

Type: string

Required: no

The Docker networking mode to use for the containers in the task. For Amazon ECS tasks hosted on Fargate, the `awsvpc` network mode is required.

When 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 [Fargate Task Networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

The `awsvpc` network mode offers the highest networking performance for containers because they use the Amazon EC2 network stack. Exposed container ports are mapped directly to the attached elastic network interface port. Because of this, you can't use dynamic host port mappings.

## Runtime platform

The following parameters are required for Fargate launch types.

`operatingSystemFamily`

Type: string

Required: Conditional

Default: LINUX

This parameter is required for Amazon ECS tasks that are hosted on Fargate.

When you register a task definition, you specify the operating system family.

The valid values for Amazon ECS tasks that are hosted on Fargate are `LINUX`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`.

The valid values for Amazon ECS tasks hosted on EC2 are `LINUX`, `WINDOWS_SERVER_2022_CORE`, `WINDOWS_SERVER_2022_FULL`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`, `WINDOWS_SERVER_2016_FULL`, `WINDOWS_SERVER_2004_CORE`, and `WINDOWS_SERVER_20H2_CORE`.

All task definitions that are used in a service must have the same value for this parameter.

When a task definition is part of a service, this value must match the service `platformFamily` value.

`cpuArchitecture`

Type: string

Required: Conditional

Default: X86\_64

This parameter is required for Amazon ECS tasks hosted on Fargate.

When you register a task definition, you specify the CPU architecture. The valid values are X86\_64 and ARM64.

All task definitions that are used in a service must have the same value for this parameter.

When you have Linux tasks for either the Fargate launch type, or the EC2 launch type, you can set the value to ARM64. For more information, see [the section called “Working with 64-bit ARM workloads on Amazon ECS” \(p. 125\)](#).

## Task size

When you register a task definition, you can specify the total CPU and memory used for the task. This is separate from the `cpu` and `memory` values at the container definition level. For tasks that are hosted on Amazon EC2 instances, these fields are optional. For tasks that are hosted on Fargate (both Linux and Windows), these fields are required and there are specific values for both `cpu` and `memory` that are supported.

### 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: conditional

### 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.

For tasks that are hosted on Fargate (both Linux and Windows containers), 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	Operating systems supported for Fargate
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB	Linux
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB	Linux
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB	Linux, Windows
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments	Linux, Windows
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments	Linux, Windows

memory

Type: string

Required: conditional

**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.

For tasks hosted on Fargate (both Linux and Windows containers), 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	Operating systems supported for Fargate
512 (0.5 GB), 1024 (1 GB), 2048 (2 GB)	256 (.25 vCPU)	Linux
1024 (1 GB), 2048 (2 GB), 3072 (3 GB), 4096 (4 GB)	512 (.5 vCPU)	Linux
2048 (2 GB), 3072 (3 GB), 4096 (4GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)	1024 (1 vCPU)	Linux, Windows
Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)	2048 (2 vCPU)	Linux, Windows
Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)	4096 (4 vCPU)	Linux, Windows

## 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 \(p. 100\)](#)
- [Advanced container definition parameters \(p. 104\)](#)
- [Other container definition parameters \(p. 115\)](#)

## Standard container definition parameters

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

**Topics**

- [Name \(p. 101\)](#)
- [Image \(p. 101\)](#)

- [Memory \(p. 101\)](#)
- [Port mappings \(p. 102\)](#)

## Name

name

Type: string

Required: yes

The name of a container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. If you're linking multiple containers in a task definition, the `name` of one container can be entered in the `links` of another container. This is to connect the containers.

## Image

image

Type: string

Required: yes

The image used to start a container. This string is passed directly to the Docker daemon. 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 \(p. 174\)](#).
- 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: conditional

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.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

#### `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 that's specified with the `memory` parameter (if applicable), or all of the available memory on the container instance, whichever comes first. This parameter maps to `MemoryReservation` in the [Create a container](#) section of the [Docker Remote API](#) and the `--memory-reservation` option to [docker run](#).

If a task-level memory value isn't specified, you must specify a non-zero integer for one or both of `memory` or `memoryReservation` in a container definition. If you specify both, `memory` must be greater than `memoryReservation`. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance 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 allows the container to only reserve 128 MiB of memory from the remaining resources on the container instance. At the same time, it also allows the container to use more memory resources when needed.

#### **Note**

This parameter is not supported for Windows containers.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container. So, don't specify less than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container. So, don't specify less than 4 MiB of memory for your containers.

## 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, only specify the `containerPort`. The `hostPort` can be left blank or it must be the same value as the `containerPort`.

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`, host ports must either be undefined or match the container port in the port mapping.

**Note**

After a task reaches the `RUNNING` status, manual and automatic host and container port assignments are visible in the following locations:

- Console: The **Network Bindings** section of a container description for a selected task.
- AWS CLI: The `networkBindings` section of the **describe-tasks** command output.
- API: The `DescribeTasks` response.

`containerPort`

Type: integer

Required: yes, when `portMappings` are used

The port number on the container that's bound to the user-specified or automatically assigned host port.

If using containers in a task with the Fargate launch type, exposed ports must be specified using `containerPort`.

For Windows containers on Fargate, you can't use port 3150 for the `containerPort`. This is because it's reserved.

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 don't 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 kept 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 ephemeral port range from 49153–65535 is used. Don't attempt to specify a host port in the ephemeral port range. This is because these are reserved for automatic assignment. In general, ports 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 might 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 that's used for the port mapping. Valid values are `tcp` and `udp`. The default is `tcp`.

If you're specifying a host port, use the following syntax.

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

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

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

## Advanced container definition parameters

The following advanced container definition parameters provide extended capabilities to the `docker run` command that's used to launch containers on your Amazon ECS container instances.

### Topics

- [Health check \(p. 104\)](#)
- [Environment \(p. 106\)](#)
- [Network settings \(p. 109\)](#)
- [Storage and logging \(p. 109\)](#)
- [Security \(p. 113\)](#)
- [Resource limits \(p. 113\)](#)
- [Docker labels \(p. 114\)](#)

## Health check

healthCheck

The container health check command and the 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 that are specified in the task definition. Amazon ECS doesn't monitor Docker health checks that are embedded in a container image but aren't specified in the container definition. Health check parameters that are specified in a container definition override any Docker health checks that exist in the container image.

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's no container health check defined.

The following describes the possible `healthStatus` values for a task. The container health check status of non-essential containers don't 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, it continues its lifecycle regardless of its health status. For tasks that are part of a service, if the task reports as unhealthy, then the task is stopped and the service scheduler replaces it.

The following are notes about container health check support:

- Container health checks are supported for Fargate tasks if you're using Linux platform version 1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 58\)](#).
- Container health checks aren't 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's healthy. The string array can start with `CMD` to run the command arguments directly, or `CMD-SHELL` to run the command with the container's default shell. If neither is specified, `CMD` is used.

When registering a task definition in the AWS Management Console, use a comma separated list of commands, which are converted to a string after the task definition is created. An example input for a health check is the following.

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

When registering a task definition using the AWS Management Console JSON panel, the AWS CLI, or the APIs, you should enclose the list of commands in brackets. An example input for a health check is the following.

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

An exit code of 0, with no `stderr` output, indicates success, and a non-zero exit code indicates failure. For more information, see `HealthCheck` in the [Create a container](#) section of the [Docker Remote API](#).

#### `interval`

The period of time (in seconds) between each health check execution. You may specify between 5 and 300 seconds. The default value is 30 seconds.

#### `timeout`

The period of time (in seconds) to wait for a health check to succeed before it's considered a failure. You 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 to provide containers time to bootstrap in before failed health checks count towards the maximum number of retries. You can specify between 0 and 300 seconds. By default, `startPeriod` is disabled.

## Environment

`cpu`

Type: integer

Required: conditional

The number of `cpu` units the Amazon ECS container agent reserves for the container. On Linux, this parameter maps to `CpuShares` in the [Create a container](#) section of the [Docker Remote API](#) and the `--cpu-shares` option to [docker run](#).

This field is optional for tasks that use the Fargate launch type. The total amount of CPU reserved for all the containers that are within a task must be lower than the task-level `cpu` value.

### Note

You can determine the number of CPU units that are available to each 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, assume that you run a single-container task on a single-core instance type with 512 CPU units specified for that container, and that task is the only task running on the container instance. In this example, the container can use the full 1,024 CPU unit share at any given time. However, assume then that you launched another copy of the same task on that container instance. Each task is guaranteed a minimum of 512 CPU units when needed, and each container can float to higher CPU usage if the other container was not using it. However, if both tasks were 100% active all of the time, they are limited to 512 CPU units.

On Linux container instances, the Docker daemon on the container instance uses the CPU value to calculate the relative CPU share ratios for running containers. 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 isn't required, and you can use CPU values below two in your container definitions. For CPU values below two (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 one are passed to Docker as one, which the Linux kernel converts to two CPU shares.
- **Agent versions >= 1.2.0:** Null, zero, and CPU values of one are passed to Docker as two CPU shares.

On Windows container instances, the CPU limit is enforced as an absolute quota. Windows containers only have access to the specified amount of CPU that's defined in the task definition. A null or zero CPU value is passed to Docker as 0, which Windows interprets as 1% of one CPU.

For additional examples, see [How Amazon ECS manages CPU and memory resources](#).

`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 doesn't affect the rest of the containers in a task. If this parameter is omitted, a container is assumed to be essential.

All tasks must have at least one essential container. If you have an application that's composed of multiple containers, 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 \(p. 90\)](#).

```
"essential": true|false
```

`entryPoint`

**Important**

Early versions of the Amazon ECS container agent don't properly handle `entryPoint` parameters. If you have problems using `entryPoint`, update your container agent or enter your commands and arguments as `command` array items instead.

Type: string array

Required: no

The entry point that's passed to the container. 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, see <https://docs.docker.com/engine/reference/builder/#entrypoint>.

```
"entryPoint": ["string", ...]
```

`command`

Type: string array

Required: no

The command that's 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, see <https://docs.docker.com/engine/reference/builder/#cmd>. If there are multiple arguments, make sure that each argument is a separated string in the array.

```
"command": ["string", ...]
```

`workingDirectory`

Type: string

Required: no

The working directory to run commands inside the container in. This parameter maps to `WorkingDir` in the [Create a container](#) section of the [Docker Remote API](#) and the `--workdir` option to [docker run](#).

```
"workingDirectory": "string"
```

#### 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 \(p. 176\)](#).

#### name

Type: String

Required: Yes

The value to set as the environment variable on the container.

#### valueFrom

Type: String

Required: Yes

The secret to expose to the container. The supported values are either the full Amazon Resource Name (ARN) of the AWS Secrets Manager secret or the full ARN of the parameter in the AWS Systems Manager Parameter Store.

##### Note

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

```
"secrets": [
  {
    "name": "environment_variable_name",
    "valueFrom": "arn:aws:ssm:region:aws_account_id:parameter/parameter_name"
  }
]
```

## Network settings

### 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 isn't supported for Windows containers or tasks using the `awsvpc` network mode.

```
"dnsServers": ["string", ...]
```

## 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 that use a log configuration, see [Example task definitions \(p. 189\)](#).

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 might use a different logging driver than the Docker daemon by specifying a log driver with this parameter in the container definition. To use a different logging driver for a container, the log system must be configured properly on the container instance (or on a different log server for remote logging options). For more information about the options for different supported log drivers, see [Configure logging drivers](#) in the Docker documentation.

Consider the following when specifying a log configuration for your containers:

- Amazon ECS supports a subset of the logging drivers that are available to the Docker daemon (shown in the valid values that follow). Additional log drivers might be available in future releases of the Amazon ECS container agent.
- This parameter requires version 1.18 or later of the Docker Remote API on your container instance.
- For tasks that use the Fargate launch type, because you don't have access to the underlying infrastructure your tasks are hosted on, any additional software needed must 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. By default, the valid values that are listed earlier are log drivers that the Amazon ECS container agent can communicate with.

For tasks that use the Fargate launch type, the supported log drivers are `awslogs`, `splunk`, and `awsfirelens`.

For more information about how to use the `awslogs` log driver in task definitions to send your container logs to CloudWatch Logs, see [Using the awslogs log driver \(p. 138\)](#).

For more information about using the `awsfirelens` log driver, see [Custom Log Routing](#).

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.



When you use FireLens to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics, you can set the `log-driver-buffer-limit` option to limit the number of events that are buffered in memory, before being sent to the log router container. It can help to resolve potential log loss issue because high throughput might result in memory running out for the buffer inside of Docker. For more information, see [the section called "Fluentd buffer limit" \(p. 146\)](#).

This parameter requires version 1.19 of the Docker Remote API or greater on your container instance.

#### `secretOptions`

Type: object array

Required: no

An object that represents the secret to pass to the log configuration. Secrets used in log configuration may include an authentication token, certificate, or encryption key, for example.) For more information, see [Specifying sensitive data \(p. 176\)](#).

##### `name`

Type: String

Required: Yes

The value to set as the environment variable on the container.

##### `valueFrom`

Type: String

Required: Yes

The secret to expose to the log configuration of the container.

```
"logConfiguration": {
  "logDriver": "splunk",
  "options": {
    "splunk-url": "https://cloud.splunk.com:8080",
    "splunk-token": "...",
    "tag": "...",
    ...
  },
  "secretOptions": [{
    "name": "splunk-token",
    "valueFrom": "/ecs/logconfig/splunkcred"
  }]
}
```

#### `firelensConfiguration`

Type: [FirelensConfiguration](#) Object

Required: No

The FireLens configuration for the container. This is used to specify and configure a log router for container logs. For more information, see [Custom log routing \(p. 144\)](#).

```
{
  "firelensConfiguration": {
    "type": "fluentd",
    "options": {
```

```

        "KeyName": ""
      }
    }
  }
}

```

options

Type: String to string map

Required: No

The options to use when configuring the log router. This field is optional and can be used to specify a custom configuration file or to add additional metadata, such as the task, task definition, cluster, and container instance details to the log event. If specified, the syntax to use is `"options":{"enable-ecs-log-metadata":"true|false", "config-file-type":"s3|file", "config-file-value":"arn:aws:s3:::mybucket/ fluent.conf | filepath"}`. For more information, see [Creating a task definition that uses a FireLens configuration](#) (p. 149).

type

Type: String

Required: Yes

The log router to use. The valid values are `fluentd` or `fluentbit`.

## Security

user

Type: string

Required: no

The user 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 specify the `user` using the following formats. If specifying a UID or GID, you must specify it as a positive integer.

- `user`
- `user:group`
- `uid`
- `uid:gid`
- `user:gid`
- `uid:group`

### Note

This parameter is not supported for Windows containers.

```
"user": "string"
```

## Resource limits

ulimits

Type: object array

Required: no

A list of `ulimit` values to define for a container. This value overwrites the default resource quota setting for the operating system. This parameter maps to `ulimits` in the [Create a container](#) section of the [Docker Remote API](#) and the `--ulimit` option to [docker run](#).

Amazon ECS tasks hosted on Fargate use the default resource limit values set by the operating system with the exception of the `nofile` resource limit parameter 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.

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 can 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 using the new console \(p. 66\)](#).

### Topics

- [Linux parameters \(p. 115\)](#)
- [Container dependency \(p. 116\)](#)
- [Container timeouts \(p. 117\)](#)
- [System controls \(p. 118\)](#)
- [Interactive \(p. 119\)](#)
- [Pseudo terminal \(p. 119\)](#)

## Linux parameters

`linuxParameters`

Type: [LinuxParameters](#) object

Required: no

Linux-specific options that are applied to the container, such as [KernelCapabilities](#).

### Note

This parameter isn't supported for Windows containers.

```
"linuxParameters": {
  "capabilities": {
    "add": ["string", ...],
    "drop": ["string", ...]
  }
}
```

`capabilities`

Type: [KernelCapabilities](#) object

Required: no

The Linux capabilities for the container that are dropped from the default configuration provided by Docker. For more information about the default capabilities and the other available capabilities, see [Runtime privilege and Linux capabilities](#) in the *Docker run reference*. For more information about these Linux capabilities, see the [capabilities\(7\)](#) Linux manual page.

add

Type: string array

Valid values: "SYS\_PTRACE"

Required: no

The Linux capabilities for the container to add to the default configuration that's provided by Docker. This parameter maps to CapAdd in the [Create a container](#) section of the [Docker Remote API](#) and the --cap-add option to [docker run](#).

drop

Type: string array

Valid values: "ALL" | "AUDIT\_CONTROL" | "AUDIT\_WRITE" | "BLOCK\_SUSPEND" | "CHOWN" | "DAC\_OVERRIDE" | "DAC\_READ\_SEARCH" | "FOWNER" | "FSETID" | "IPC\_LOCK" | "IPC\_OWNER" | "KILL" | "LEASE" | "LINUX\_IMMUTABLE" | "MAC\_ADMIN" | "MAC\_OVERRIDE" | "MKNOD" | "NET\_ADMIN" | "NET\_BIND\_SERVICE" | "NET\_BROADCAST" | "NET\_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS\_ADMIN" | "SYS\_BOOT" | "SYS\_CHROOT" | "SYS\_MODULE" | "SYS\_NICE" | "SYS\_PACCT" | "SYS\_PTRACE" | "SYS\_RAWIO" | "SYS\_RESOURCE" | "SYS\_TIME" | "SYS\_TTY\_CONFIG" | "SYSLOG" | "WAKE\_ALARM"

Required: no

The Linux capabilities for the container to remove from the default configuration that's provided by Docker. This parameter maps to CapDrop in the [Create a container](#) section of the [Docker Remote API](#) and the --cap-drop option to [docker run](#).

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.

## 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 \(p. 192\)](#).

### Note

If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

For Amazon ECS tasks that are hosted on Fargate, this parameter requires that the task or service uses platform version 1.3.0 or later (Linux) or 1.0.0 (Windows).

```
"dependsOn": [
```

```
{
  "containerName": "string",
  "condition": "string"
}
```

`containerName`

Type: String

Required: Yes

The container name that must meet the specified condition.

`condition`

Type: String

Required: Yes

The dependency condition of the container. The following are the available conditions and their behavior:

- **START** – This condition emulates the behavior of links and volumes today. 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 non-essential containers that run a script and then exit. This condition can't be set on an essential container.
- **SUCCESS** – This condition is the same as **COMPLETE**, but it also requires that the container exits with a zero status. This condition can't be set on an essential container.
- **HEALTHY** – This condition validates that the dependent container passes its 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 doesn't reach the desired status within that time, then `containerA` doesn't start.

### Note

If a container doesn't meet a dependency constraint or times out before meeting the constraint, Amazon ECS doesn't progress dependent containers to their next state.

For Amazon ECS tasks that are hosted on Fargate, this parameter requires that the task or service uses platform version 1.3.0 or later (Linux).

`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 that use the Fargate launch type, the task or service requires platform version 1.3.0 or later (Linux) or 1.0.0 or later (for Windows). The max stop timeout value is 120 seconds. However, if the parameter isn't specified, the default value of 30 seconds is used.

## System controls

`systemControls`

Type: [SystemControl](#) object

Required: no

A list of namespaced kernel parameters to set in the container. This parameter maps to `Sysctl`s in the [Create a container](#) section of the [Docker Remote API](#) and the `--sysctl` option to [docker run](#).

We do not recommend that you specify network-related `systemControls` parameters for multiple containers in a single task that also uses either the `awsvpc` or `host` network mode for the following reasons:

- For tasks that use the `awsvpc` network mode, if you set `systemControls` for any container, it applies to all containers in the task. If you set different `systemControls` for multiple containers in a single task, the container that's started last determines which `systemControls` take effect.
- For tasks that use the `host` network mode, the network namespace `systemControls` aren't supported.

If you're setting an IPC resource namespace to use for the containers in the task, the following conditions apply to your system controls. For more information, see [IPC mode \(p. 123\)](#).

- For tasks that use the `host` IPC mode, IPC namespace `systemControls` aren't supported.
- For tasks that use the `task` IPC mode, IPC namespace `systemControls` values applies 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 `Sysctl`s beginning with `"fs.mqueue.*"`

Valid network namespace values: Sysctls beginning with "net.\*"  
value

Type: String

Required: no

The value for the namespaced kernel parameter that's specified in namespace.

## Interactive

interactive

Type: Boolean

Required: no

When this parameter is `true`, you can deploy containerized applications that require stdin or a tty to be allocated. This parameter maps to `OpenStdin` in the [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](#).

# Proxy configuration

proxyConfiguration

Type: [ProxyConfiguration](#) object

Required: no

The configuration details for the App Mesh proxy.

For tasks that use 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 serves as the App Mesh proxy.

`properties`

Type: Array of [KeyValuePair](#) objects

Required: No

The set of network configuration parameters to provide the Container Network Interface (CNI) plugin, specified as key-value pairs.

- `IgnoredUID` – (Required) The user ID (UID) of the proxy container as defined by the user parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredGID` is specified, this field can be empty.
- `IgnoredGID` – (Required) The group ID (GID) of the proxy container as defined by the user parameter in a container definition. This is used to ensure the proxy ignores its own traffic. If `IgnoredUID` is specified, this field can be empty.
- `AppPorts` – (Required) The list of ports that the application uses. Network traffic to these ports is forwarded to the `ProxyIngressPort` and `ProxyEgressPort`.
- `ProxyIngressPort` – (Required) Specifies the port that incoming traffic to the `AppPorts` is directed to.
- `ProxyEgressPort` – (Required) Specifies the port that outgoing traffic from the `AppPorts` is directed to.
- `EgressIgnoredPorts` – (Required) The outbound traffic going to these specified ports is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.
- `EgressIgnoredIPs` – (Required) The outbound traffic going to these specified IP addresses is ignored and not redirected to the `ProxyEgressPort`. It can be an empty list.

`name`

Type: String

Required: No

The name of the key-value pair.

`value`

Type: String

Required: No

The value of the key-value pair.

## Volumes

When you register a task definition, you can optionally specify a list of volumes to be passed to the Docker daemon on a container instance, which then becomes available for access by other containers on the same container instance.

For more information, see [Using data volumes in tasks \(p. 127\)](#).

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` object.

`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.

### **Important**

If an EFS access point is specified in the `authorizationConfig`, the `rootDirectory` parameter must either be omitted or set to `/` which will enforce the path set on the EFS access point.

`transitEncryption`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

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 in the `efsVolumeConfiguration` must either be omitted or set to `/` which will enforce the path set on the EFS access point. If an access point is used, transit encryption must be enabled in the `efsVolumeConfiguration`. For more information, see [Working with Amazon EFS Access Points](#) in the *Amazon Elastic File System User Guide*.

`iam`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

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](#).

## Tags

When you register a task definition, you can optionally specify metadata tags that are applied to the task definition. Tags help you categorize and organize your task definition. Each tag consists of a key and an optional value. You define both of them. For more information, see [Tagging your Amazon ECS resources](#) (p. 276).

### Important

Don't add personally identifiable information or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags aren't intended to be used for private or sensitive data.

The following parameters are allowed in a tag object.

key

Type: string

Required: no

One part of a key-value pair that make up a tag. A key is a general label that acts like a category for more specific tag values.

value

Type: string

Required: no

The optional part of a key-value pair that make up a tag. A value acts as a descriptor within a tag category (key).

## Other task definition parameters

The following task definition parameters can be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

### Topics

- [Ephemeral storage \(p. 123\)](#)
- [IPC mode \(p. 123\)](#)
- [PID mode \(p. 124\)](#)

## Ephemeral storage

ephemeralStorage

Type: Object

Required: No

The amount of ephemeral storage (in GB) to allocate for the task. This parameter is used to expand the total amount of ephemeral storage available, beyond the default amount, for tasks that are hosted on AWS Fargate. For more information, see [the section called "Bind mounts" \(p. 132\)](#).

### Note

This parameter is only supported for tasks that are hosted on AWS Fargate using platform version 1.4.0 or later (Linux). This isn't supported for Windows containers on Fargate.

## IPC mode

ipcMode

Type: String

Required: No

The IPC resource namespace to use for the containers in the task. The valid values are `host`, `task`, or `none`. If `host` is specified, then all the containers that are within the tasks that specified the `host`

IPC mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If `task` is specified, all the containers that are within the specified task share the same IPC resources. If `none` is specified, then IPC resources within the containers of a task are private and not shared with other containers in a task or on the container instance. If no value is specified, then the value for `ipcMode` is set to `shareable`. For more information, see [IPC settings](#) in the *Docker run reference*.

If the `host` IPC mode is used, there's a heightened risk of undesired IPC namespace exposure. For more information, see [Docker security](#).

If you're setting namespaced kernel parameters that use `systemControls` for the containers in the task, the following applies to your IPC resource namespace. For more information, see [System controls \(p. 118\)](#).

- For tasks that use the `host` IPC mode, IPC namespace that's related `systemControls` aren't supported.
- For tasks that use the `task` IPC mode, `systemControls` that relate to the IPC namespace apply to all containers within a task.

**Note**

This parameter is not supported for Windows containers or tasks using the Fargate launch type.

## PID mode

`pidMode`

Type: String

Required: No

The process namespace to use for the containers in the task. The valid values are `host` or `task`. If `host` is specified, all containers within the tasks that specified the `host` PID mode on the same container instance share the same process namespace with the host Amazon EC2 instance. If `task` is specified, all containers within the specified task share the same process namespace. If no value is specified, the default is a private namespace. For more information, see [PID settings](#) in the *Docker run reference*.

If the `host` PID mode is used, there's 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

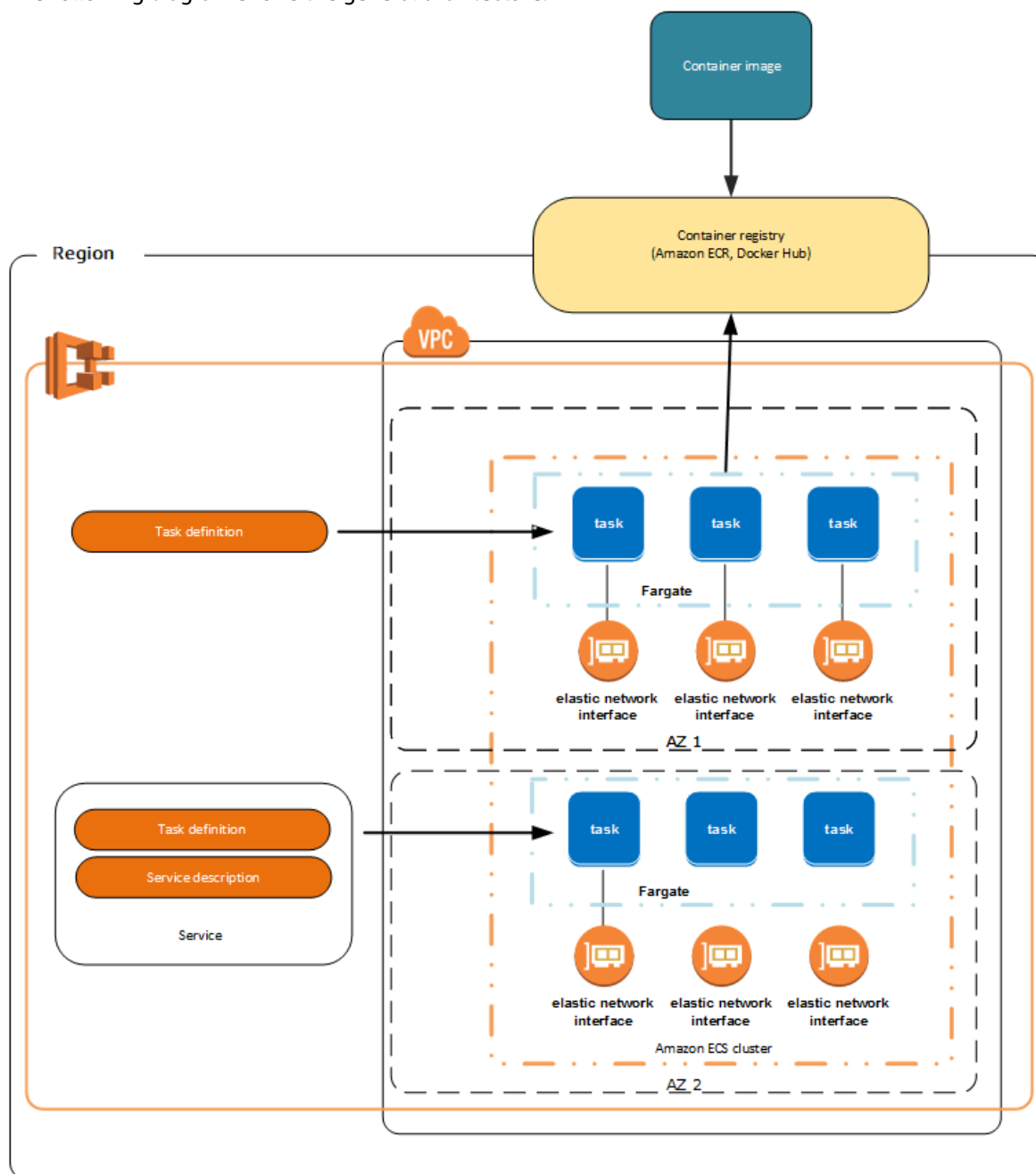
You can specify an Amazon ECS launch type when you run a standalone task or create a service. Doing so determines the infrastructure that your tasks and services are hosted on. The following are the available launch types.

### Fargate launch type

You can use the Fargate launch type to run your containerized applications without the need of provisioning and managing the underlying infrastructure. AWS Fargate is the serverless way to host your Amazon ECS workloads.

For information about the Regions that support Fargate, see [the section called “AWS Fargate Regions”](#) (p. 285).

The following diagram shows the general architecture.



## Working with 64-bit ARM workloads on Amazon ECS

Amazon ECS supports using 64-bit ARM applications. You can run your applications on the platform that's powered by [AWS Graviton2](#) processors. It's suitable for a wide variety of workloads. This includes

workloads such as application servers, micro-services, high-performance computing, CPU-based machine learning inference, video encoding, electronic design automation, gaming, open-source databases, and in-memory caches.

## Considerations

Before you begin deploying task definitions that use the 64-bit ARM architecture, consider the following:

- The applications can use the Fargate or EC2 launch types.
- The applications can only use the Linux operating system.
- For the Fargate type, the applications must use Fargate platform version 1.4.0 or later.
- The applications can use Fluent Bit or CloudWatch for monitoring.
- For the Fargate launch type, the following AWS Regions do not support 64-bit ARM workloads:
  - US East (N. Virginia), the `us-east-1-az3` Availability Zone
  - China (Beijing)
  - China (Ningxia)
  - Africa (Cape Town)
  - Middle East (Bahrain)
  - AWS GovCloud (US-East)
  - AWS GovCloud (US-West)
  - In Asia Pacific (Osaka), specifically the `apne3-az2` and `apne3-az3` Availability Zones only
- For the Amazon EC2 launch type, see the following to verify that the Region that you're in supports the instance type you want to use:
  - [Amazon EC2 M6g Instances](#)
  - [Amazon EC2 T4g Instances](#)
  - [Amazon EC2 C6g Instances](#)
  - [Amazon EC2 R6gd Instances](#)
  - [Amazon EC2 X2gd Instances](#)

You can also use the Amazon EC2 `describe-instance-type-offerings` command with a filter to view the instance offering for your Region.

```
aws ec2 describe-instance-type-offerings --filters Name=instance-type,Values=instance-type --region region
```

The following example checks for the M6 instance type availability in the US East (N. Virginia) (`us-east-1`) Region.

```
aws ec2 describe-instance-type-offerings --filters Name=instance-type,Values=M6 --region us-east-1
```

For more information, see [describe-instance-type-offerings](#) in the *Amazon EC2 Command Line Reference*.

## Specifying the ARM architecture in your task definition

---

To use the ARM architecture, specify `ARM64` for the `cpuArchitecture` task definition parameter.

In the following example, the ARM architecture is specified in a task definition. It's in JSON format.

```
{
  "runtimePlatform": {
    "operatingSystemFamily": "LINUX",
    "cpuArchitecture": "ARM64"
  },
  ...
}
```

In the following example, a task definition for the ARM architecture displays "hello world."

```
{
  "family": "arm64-testapp",
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "arm-container",
      "image": "arm64v8/busybox",
      "cpu": 100,
      "memory": 100,
      "essential": true,
      "command": [ "echo hello world" ],
      "entryPoint": [ "sh", "-c" ]
    }
  ],
  "requiresCompatibilities": [ "FARGATE" ],
  "cpu": "256",
  "memory": "512",
  "runtimePlatform": {
    "operatingSystemFamily": "LINUX",
    "cpuArchitecture": "ARM64"
  },
  "executionRoleArn": "arn:aws:iam::123456789012:role/ec2TaskExecutionRole"
}
```

## Interfaces for configuring ARM

You can configure the ARM CPU architecture for Amazon ECS task definitions using one of the following interfaces:

- New Amazon ECS console
- AWS Command Line Interface (AWS CLI)
- AWS SDKs
- AWS Copilot

## Using data volumes in tasks

Amazon ECS on Fargate supports the following data volume options for containers.

- Amazon EFS volumes — Provides simple, scalable, and persistent file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic. It grows and shrinks automatically as you add and remove files. Your applications can have the storage that they need and when they need it. For more information, see [Amazon EFS volumes \(p. 129\)](#).
- Bind mounts — A file or directory on the host, such as AWS Fargate, is mounted into a container. For more information, see [Bind mounts \(p. 132\)](#).



## Fargate task storage

When provisioned, each Amazon ECS task hosted on Linux containers on AWS Fargate receives the following ephemeral storage for bind mounts. This can be mounted and shared among containers that use the `volumes`, `mountPoints`, and `volumesFrom` parameters in the task definition. This isn't supported for Windows containers on AWS Fargate.

## Fargate Linux container platform versions

### Fargate tasks using platform version 1.4.0 or later

By default, Amazon ECS tasks that are hosted on Fargate using platform version 1.4.0 or later receive a minimum of 20 GiB of ephemeral storage. The total amount of ephemeral storage can be increased, up to a maximum of 200 GiB. You can do this by specifying the `ephemeralStorage` parameter in your task definition.

The pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage your task has to use, you must subtract the amount of storage your container image uses from the total amount of ephemeral storage your task is allocated.

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

### Fargate tasks using platform version 1.3.0 or earlier

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

- 10 GB of Docker layer storage

**Note**

This amount includes both compressed and uncompressed container image artifacts.

- An additional 4 GB for volume mounts. This can be mounted and shared among containers that use the `volumes`, `mountPoints`, and `volumesFrom` parameters in the task definition.

## Fargate Windows container platform versions

### Fargate tasks using platform version 1.0.0 or later

By default, Amazon ECS tasks that are hosted on Fargate using platform version 1.0.0 or later receive a minimum of 20 GiB of ephemeral storage.

The pulled, compressed, and the uncompressed container image for the task is stored on the ephemeral storage. To determine the total amount of ephemeral storage your task has to use, you must subtract the amount of storage your container image uses from the total amount of ephemeral storage your task is allocated.

The ephemeral storage is encrypted with an AES-256 encryption algorithm, which uses an AWS owned encryption key.

For more information, see [Bind mounts \(p. 132\)](#).

#### Topics

- [Amazon EFS volumes \(p. 129\)](#)
- [Bind mounts \(p. 132\)](#)

## Amazon EFS volumes

Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic. It grows and shrinks automatically as you add and remove files. Your applications can have the storage they need and when they need it.

You can use Amazon EFS file systems with Amazon ECS to export file system data across your fleet of container instances. That way, your tasks have access to the same persistent storage, no matter the instance on which they land. Your task definitions must reference volume mounts on the container instance to use the file system. The following sections describe how to get started using Amazon EFS with Amazon ECS.

### Amazon EFS volume considerations

Consider the following when using Amazon EFS volumes:

- For tasks that are hosted on Fargate, Amazon EFS file systems are supported on platform version 1.4.0 or later (Linux). For more information, see [AWS Fargate platform versions \(p. 58\)](#).
- When using Amazon EFS volumes for tasks that are hosted on Fargate, Fargate creates a supervisor container that's responsible for managing the Amazon EFS volume. The supervisor container uses a small amount of the task's memory. The supervisor container is visible when querying the task metadata version 4 endpoint. However, it isn't visible in CloudWatch Container Insights. For more information, see [Task metadata endpoint version 4 \(p. 380\)](#).
- Using Amazon EFS volumes or specifying an `EFSVolumeConfiguration` isn't supported on external instances.

### Using Amazon EFS access points

Amazon EFS access points are application-specific entry points into an EFS file system for managing application access to shared datasets. For more information about Amazon EFS access points and how to control access to them, see [Working with Amazon EFS Access Points](#) in the *Amazon Elastic File System User Guide*.

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

#### Note

When creating an EFS access point, specify a path on the file system to serve as the root directory. When referencing the EFS file system with an access point ID in your Amazon ECS task definition, the root directory must either be omitted or set to `/`, which enforces the path set on the EFS access point.

You can use an Amazon ECS task IAM role to enforce that specific applications use a specific access point. By combining IAM policies with access points, you can provide secure access to specific datasets for your applications. For more information about how to use task IAM roles, see [IAM roles for tasks \(p. 360\)](#).

### Specifying an Amazon EFS file system in your task definition

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

```
{
  "containerDefinitions": [
    {
      "name": "container-using-efs",
      "image": "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 is used. Specifying / has the same effect as omitting this parameter.

### Important

If an EFS access point is specified in the `authorizationConfig`, the root directory parameter must either be omitted or set to `/`, which enforces the path set on the EFS access point.

#### `transitEncryption`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. If Amazon EFS IAM authorization is used, transit encryption must be enabled. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see [Encrypting Data in Transit](#) in the *Amazon Elastic File System User Guide*.

#### `transitEncryptionPort`

Type: Integer

Required: No

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

#### `authorizationConfig`

Type: Object

Required: No

The authorization configuration details for the Amazon EFS file system.

#### `accessPointId`

Type: String

Required: No

The access point ID to use. If an access point is specified, the root directory value in the `efsVolumeConfiguration` must either be omitted or set to `/`, which enforces the path set on the EFS access point. If an access point is used, transit encryption must be enabled in the `efsVolumeConfiguration`. For more information, see [Working with Amazon EFS Access Points](#) in the *Amazon Elastic File System User Guide*.

#### `iam`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Specifies whether to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, transit encryption must be enabled in the `efsVolumeConfiguration`. If this parameter is omitted, the default value of `DISABLED` is used. For more information, see [IAM Roles for Tasks](#).

## Bind mounts

With bind mounts, a file or directory on a host, such as AWS Fargate, is mounted into a container. Bind mounts are tied to the lifecycle of the container that uses them. After all of the containers that use a bind mount are stopped, such as when a task is stopped, the data is removed. For more information, see [Using bind mounts](#) in the Docker documentation.

The following are common use cases for bind mounts.

- To provide an empty data volume to mount in one or more containers.
- To expose a path and its contents from a Dockerfile to one or more containers.

## Considerations when using bind mounts

When using bind mounts, consider the following.

- For tasks that are hosted on AWS Fargate using platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows), by default they receive a minimum of 20 GiB of ephemeral storage for bind mounts. The total amount of ephemeral storage can be increased to a maximum of 200 GiB by specifying the `ephemeralStorage` object in your task definition.
- To expose files from a Dockerfile to a data volume when a task is run, the Amazon ECS data plane looks for a `VOLUME` directive. If the absolute path that's specified in the `VOLUME` directive is the same as the `containerPath` that's specified in the task definition, the data in the `VOLUME` directive path is copied to the data volume. In the following Dockerfile example, a file that's named `examplefile` in the `/var/log/exported` directory is written to the host and then mounted inside the container.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN mkdir -p /var/log/exported
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]
```

By default, the volume permissions are set to 0755 and the owner as `root`. You can customize these permissions in the Dockerfile. The following example defines the owner of the directory as `node`.

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
RUN touch /var/log/exported/examplefile
USER node
VOLUME ["/var/log/exported"]
```

## Specifying a bind mount in your task definition

For Amazon ECS tasks that are hosted on Fargate, the following task definition JSON snippet shows the syntax for the `volumes`, `mountPoints`, and `ephemeralStorage` objects for a task definition.

```
{
  "family": "",
  ...
  "containerDefinitions" : [
    {
      "mountPoints" : [
        {
          "containerPath" : "/path/to/mount_volume",
```

```

        "sourceVolume" : "string"
      }
    ],
    "name" : "string"
  }
],
...
"volumes" : [
  {
    "name" : "string"
  }
],
"ephemeralStorage": {
  "sizeInGiB": integer
}
}

```

The following describes each task definition parameter in more detail.

#### name

Type: String

Required: No

The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. This name is referenced in the `sourceVolume` parameter of container definition `mountPoints`.

#### 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`.

`ephemeralStorage`

Type: Object

Required: No

The amount of ephemeral storage to allocate for the task. This parameter is used to expand the total amount of ephemeral storage available, beyond the default amount, for tasks hosted on AWS Fargate using platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows).

You can use the Copilot CLI, CloudFormation, the AWS SDK or the CLI to specify ephemeral storage for a bind mount.

## Bind mount examples

The following examples cover the most common use cases for using a bind mount for your containers.

### To allocate an increased amount of ephemeral storage space for a Fargate task

For Amazon ECS tasks that are hosted on Fargate using platform version 1.4.0 or later (Linux) or 1.0.0 (Windows), you can allocate more than the default amount of ephemeral storage for the containers in your task to use. This example can be incorporated into the other examples to allocate more ephemeral storage for your Fargate tasks.

- In the task definition, define an `ephemeralStorage` object. The `sizeInGiB` must be an integer between the values of 21 and 200 and is expressed in GiB.

```
"ephemeralStorage": {  
  "sizeInGiB": integer  
}
```

### To provide an empty data volume for one or more containers

In some cases, you want to provide the containers in a task some scratch space. For example, you might have two database containers that need to access the same scratch file storage location during a task. This can be achieved using a bind mount.

1. In the task definition `volumes` section, define a bind mount with the name `database_scratch`.

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

2. In the `containerDefinitions` section, create the database container definitions. This is so that they mount the volume.

```
"containerDefinitions": [  
  {  
    "name": "database1",  
    "image": "my-repo/database",  
    "cpu": 100,  
    "memory": 100,  
    "essential": true,  
  },  
  {  
    "name": "database2",  
    "image": "my-repo/database",  
    "cpu": 100,  
    "memory": 100,  
    "essential": true,  
  }  
]
```

```

    "mountPoints": [
      {
        "sourceVolume": "database_scratch",
        "containerPath": "/var/scratch"
      }
    ],
  },
  {
    "name": "database2",
    "image": "my-repo/database",
    "cpu": 100,
    "memory": 100,
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "database_scratch",
        "containerPath": "/var/scratch"
      }
    ]
  }
]

```

### To expose a path and its contents in a Dockerfile to a container

In this example, you have a Dockerfile that writes data that you want to mount inside a container.

1. Create a Dockerfile. The following example uses the public Amazon Linux 2 container image and creates a file that's named `examplefile` in the `/var/log/exported` directory that we want to mount inside the container. The `VOLUME` directive should specify an absolute path.

```

FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN mkdir -p /var/log/exported
RUN touch /var/log/exported/examplefile
VOLUME ["/var/log/exported"]

```

By default, the volume permissions are set to `0755` and the owner as `root`. These permissions can be changed in the Dockerfile. In the following example, the owner of the `/var/log/exported` directory is set to `node`.

```

FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install -y shadow-utils && yum clean all
RUN useradd node
RUN mkdir -p /var/log/exported && chown node:node /var/log/exported
RUN touch /var/log/exported/examplefile
USER node
VOLUME ["/var/log/exported"]

```

2. In the task definition volumes section, define a volume with the name `application_logs`.

```

"volumes": [
  {
    "name": "application_logs",
  }
]

```

3. In the `containerDefinitions` section, create the application container definitions. This is so they mount the storage. The `containerPath` value must match the absolute path that's specified in the `VOLUME` directive from the Dockerfile.

```

"containerDefinitions": [

```



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

## Fargate task networking

### Important

If you're using Amazon ECS tasks that are hosted on Amazon EC2 instances, see [Task networking](#) in the *Amazon Elastic Container Service Developer Guide* for networking information that's relevant to your instances.

By default, every Amazon ECS task on Fargate is provided an elastic network interface (ENI) with a primary private IP address. When using a public subnet, you can optionally assign a public IP address to the task's ENI. If your VPC is enabled for dual-stack mode and you use a subnet with an IPv6 CIDR block, your task's ENI also receives an IPv6 address. A task can only have one ENI that's associated with it at a time. Containers that belong to the same task can also communicate over the `localhost` interface. For more information about VPCs and subnets, see [VPCs and subnets](#) in the *Amazon VPC User Guide*.

For a task on Fargate to pull a container image, the task must have a route to the internet. The following describes how you can verify that your task has a route to the internet.

- When using a public subnet, you can assign a public IP address to the task ENI.
- When using a private subnet, the subnet can have a NAT gateway attached.
- When using container images that are hosted in Amazon ECR, you can configure Amazon ECR to use an interface VPC endpoint and the image pull occurs over the task's private IPv4 address. For more information, see [Amazon ECR interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon Elastic Container Registry User Guide*.

Because each task gets its own ENI, you can use networking features such as VPC Flow Logs, which you can use to monitor traffic to and from your tasks. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You can also take advantage of AWS PrivateLink. You can configure an VPC interface endpoint so that you can access Amazon ECS APIs through private IP addresses. AWS PrivateLink restricts all network

traffic between your VPC and Amazon ECS to the Amazon network. You don't need an internet gateway, a NAT device, or a virtual private gateway. For more information, see [AWS PrivateLink](#) in the *Amazon ECS Best Practices Guide*.

The ENIs that are created are fully managed by AWS Fargate. Moreover, there's an associated IAM policy that's used to grant permissions for Fargate. For tasks using Fargate platform version 1.4.0 or later, the task receives a single ENI (referred to as the task ENI) and all network traffic flows through that ENI within your VPC. This traffic is recorded in your VPC flow logs. For tasks that use Fargate platform version 1.3.0 and earlier, in addition to the task ENI, the task also receives a separate Fargate owned ENI, which is used for some network traffic that isn't visible in the VPC flow logs. The following table describes the network traffic behavior and the required IAM policy for each platform version.

Action	Traffic flow with Linux platform version 1.3.0 and earlier	Traffic flow with Linux platform version 1.4.0	Traffic flow with Windows platform version 1.0.0	IAM permission
Retrieving Amazon ECR login credentials	Fargate owned ENI	Task ENI	Task ENI	Task execution IAM role
Image pull	Task ENI	Task ENI	Task ENI	Task execution IAM role
Sending logs through a log driver	Task ENI	Task ENI	Task ENI	Task execution IAM role
Sending logs through FireLens for Amazon ECS	Task ENI	Task ENI	Task ENI	Task IAM role
Retrieving secrets from Secrets Manager or Systems Manager	Fargate owned ENI	Task ENI	Task ENI	Task execution IAM role
Amazon EFS file system traffic	Not available	Task ENI	Task ENI	Task IAM role
Application traffic	Task ENI	Task ENI	Task ENI	Task IAM role

## Fargate task networking considerations

Consider the following when using task networking.

- The Amazon ECS service-linked role is required to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you when you create a cluster or if you create or update a service in the AWS Management Console. For more information, see [Service-linked role for Amazon ECS \(p. 348\)](#). You can also create the service-linked role using the following AWS CLI command.

```
aws iam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

- Amazon ECS populates the hostname of the task with an Amazon provided DNS hostname when both the `enableDnsHostnames` and `enableDnsSupport` options are enabled on your VPC. If these options aren't enabled, the DNS hostname of the task is set to a random hostname. For more

information about the DNS settings for a VPC, see [Using DNS with Your VPC](#) in the *Amazon VPC User Guide*.

- You can only specify up to 16 subnets and 5 security groups for `awsVpcConfiguration`. For more information, see [AwsVpcConfiguration](#) in the *Amazon Elastic Container Service API Reference*.
- You can't manually detach or modify the ENIs that are created and attached by Fargate. This is to prevent the accidental deletion of an ENI that's associated with a running task. To release the ENIs for a task, stop the task.
- If a VPC subnet is updated to change the DHCP options set it uses, you can't also apply these changes to existing tasks that use the VPC. Start new tasks, which will receive the new setting to smoothly migrate while testing the new change and then stop the old ones, if no rollback is required.
- Tasks that are launched in subnets with IPv6 CIDR blocks only receive an IPv6 address when using Fargate platform version 1.4.0 or later for Linux or 1.0.0 for Windows.
- For tasks that use platform version 1.4.0 or later for Linux or 1.0.0 for Windows, the task ENIs support jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames reduces overhead when the network path between your task and the destination supports jumbo frames.
- Services with tasks that use the Fargate launch type only support Application Load Balancer and Network Load Balancer. Classic Load Balancer isn't supported. When you create any target groups, you must choose `ip` as the target type, not `instance`. For more information, see [Service load balancing](#) (p. 252).

## Using a VPC in dual-stack mode

When using a VPC in dual-stack mode, your tasks can communicate over IPv4 or IPv6, or both. IPv4 and IPv6 addresses are independent of each other and you must configure routing and security in your VPC separately for IPv4 and IPv6. For more information about configuring your VPC for dual-stack mode, see [Migrating to IPv6](#) in the *Amazon VPC User Guide*.

If the following conditions are met, Amazon ECS tasks on Fargate are assigned an IPv6 address:

- Your VPC and subnet are enabled for IPv6. For more information about how to configure your VPC for dual-stack mode, see [Migrating to IPv6](#) in the *Amazon VPC User Guide*.
- The task or service uses Fargate platform version 1.4.0 or later for Linux or 1.0.0 for Windows.
- The `dualStackIPv6` account setting is enabled. For more information, see [Account settings](#) (p. 196).

If you configure your VPC with an internet gateway or an outbound-only internet gateway, Amazon ECS tasks on Fargate that are assigned an IPv6 address can access the internet. NAT gateways aren't needed. For more information, see [Internet gateways](#) and [Egress-only internet gateways](#) in the *Amazon VPC User Guide*.

## Using the awslogs log driver

You can configure the containers in your tasks to send log information to CloudWatch Logs. If you do this, you can view the logs from the containers in your Fargate tasks. This topic goes over how you can 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 typically might see in an interactive terminal if you ran the container locally, which are the `STDOUT` and `STDERR` I/O streams. The `awslogs` log driver simply passes these logs from Docker to CloudWatch Logs. For more information about how Docker logs are processed, including alternative ways to capture different file data or streams, see [View logs for a container or service](#) in the Docker documentation.

## Turning on the awslogs log driver for your containers

If you're using the Fargate launch type for your tasks, you need to add the required `logConfiguration` parameters to your task definition to turn on the `awslogs` log driver. For more information, see [Specifying a log configuration in your task definition](#) (p. 141).

## Creating a log group

The `awslogs` log driver can send log streams to an existing log group in CloudWatch Logs or 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 creates 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.

The following code shows how to set the `awslogs-create-group` option.

```
{
  "containerDefinitions": [
    {
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "firelens-container",
          "awslogs-region": "us-west-2",
          "awslogs-create-group": "true",
          "awslogs-stream-prefix": "firelens"
        }
      }
    }
  ]
}
```

## Using the auto-configuration feature to create a log group

When registering a task definition in the Amazon ECS console, you can allow Amazon ECS to auto-configure your CloudWatch logs. Doing this causes a log group to be created 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. 141).

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 to be created automatically. If this option isn't specified, it defaults to `false`.

#### **Note**

Your IAM policy must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`.

### `awslogs-region`

Required: Yes

Specify the AWS Region that the `awslogs` log driver is to send your Docker logs to. You can choose to send all of your logs from clusters in different Regions to a single region in CloudWatch Logs. This is so that they're all visible in one location. Otherwise, you can separate them by Region for more granularity. Make sure that the specified log group exists in the Region that you specify with this option.

### `awslogs-group`

Required: Yes

Make sure to specify a log group that the `awslogs` log driver sends its log streams to. For more information, see [Creating a log group \(p. 139\)](#).

### `awslogs-stream-prefix`

Required: Yes, when using the Fargate launch type.

Use the `awslogs-stream-prefix` option to associate a log stream with the specified prefix, the container name, and the ID of the Amazon ECS task that the container belongs to. If you specify a prefix with this option, then the log stream takes the following format.

```
prefix-name/container-name/ecs-task-id
```

For Amazon ECS services, you can use the service name as the prefix. Doing so you can trace log streams to the service that the container belongs to, the name of the container that sent them, and the ID of the task that the container belongs to.

### `awslogs-datetime-format`

Required: No

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

One example of a use case for using this format is for parsing output such as a stack dump, which might otherwise be logged in multiple entries. The correct pattern allows it to be captured in a single entry.

For more information, see [awslogs-datetime-format](#).

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. This might have a negative impact on logging performance.

`awslogs-multiline-pattern`

Required: No

This option defines a multiline start pattern that uses a regular expression. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. The matched line is the delimiter between log messages.

For more information, see [awslogs-multiline-pattern](#).

This option is ignored if `awslogs-datetime-format` is also configured.

**Note**

Multiline logging performs regular expression parsing and matching of all log messages. This might have a negative impact on logging performance.

`mode`

Required: No

Valid values: `non-blocking` | `blocking`

Default value: `blocking`

The delivery mode of log messages from the container to `awslogs`. For more information, see [Configure logging drivers](#).

`max-buffer-size`

Required: No

Default value: `1m`

When `non-blocking` mode is used, the `max-buffer-size` log option controls the size of the ring buffer that's used for intermediate message storage.

## 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 using the new console \(p. 66\)](#).

The task definition JSON that follows has a `logConfiguration` object specified for each container. One is for the WordPress container that sends logs to a log group called `awslogs-wordpress`. The other is for a MySQL container that sends logs to a log group that's called `awslogs-mysql`. Both containers use the `awslogs-example` log stream prefix.

```
{
  "containerDefinitions": [
    {
      "name": "wordpress",
      "links": [
        "mysql"
      ]
    }
  ]
}
```

```
    ],
    "image": "wordpress",
    "essential": true,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "logConfiguration": {
      "logDriver": "awslogs",
      "options": {
        "awslogs-create-group": "true",
        "awslogs-group": "awslogs-wordpress",
        "awslogs-region": "us-west-2",
        "awslogs-stream-prefix": "awslogs-example"
      }
    },
    "memory": 500,
    "cpu": 10
  },
  {
    "environment": [
      {
        "name": "MYSQL_ROOT_PASSWORD",
        "value": "password"
      }
    ],
    "name": "mysql",
    "image": "mysql",
    "cpu": 10,
    "memory": 500,
    "essential": true,
    "logConfiguration": {
      "logDriver": "awslogs",
      "options": {
        "awslogs-create-group": "true",
        "awslogs-group": "awslogs-mysql",
        "awslogs-region": "us-west-2",
        "awslogs-stream-prefix": "awslogs-example"
      }
    }
  }
],
"family": "awslogs-example"
}
```

## Viewing awslogs container logs in CloudWatch Logs

After your Fargate tasks that use the awslogs log driver have launched, 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.

## Log Configuration

Log driver: awslogs [View logs in CloudWatch](#)

Key	Value
awslogs-group	awslogs-wordpress
awslogs-region	ap-northeast-1
awslogs-stream-prefix	awslogs-example

**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. 139).

The screenshot shows the AWS CloudWatch console interface. At the top, there are two buttons: "Create Metric Filter" (blue) and "Actions" (grey with a dropdown arrow). Below these is a "Filter:" section with a text input containing "Log Group Name Prefix" and a clear "x" button. Underneath the filter is a section titled "Log Groups" with a list of log groups. Two log groups are visible: "awslogs-mysql" and "awslogs-wordpress", each preceded by an unchecked checkbox.

<b>Create Metric Filter</b>	<b>Actions</b> ▼
<b>Filter:</b> Log Group Name Prefix x	
<b>Log Groups</b>	
<input type="checkbox"/>	awslogs-mysql
<input type="checkbox"/>	awslogs-wordpress

4. Choose a log stream to view.



Filter events		
	Time (UTC -07:00)	Message
2016-09-09		
No older events found at the		
▶	12:56:47	WordPress not found in /var/www/html -
▶	12:56:47	Complete! WordPress has been success
▶	12:56:49	AH00558: apache2: Could not reliably d
▶	12:56:49	AH00558: apache2: Could not reliably d
▶	12:56:49	[Fri Sep 09 19:56:49.059245 2016] [mpm
▶	12:56:49	[Fri Sep 09 19:56:49.059273 2016] [core
▶	13:06:55	52.90.111.181 - - [09/Sep/2016:20:06:55
▶	13:06:56	52.90.111.181 - - [09/Sep/2016:20:06:55
▶	13:06:56	52.90.111.181 - - [09/Sep/2016:20:06:56
▶	13:06:57	54.210.246.190 - - [09/Sep/2016:20:06:5

## Custom log routing

You can use FireLens for Amazon ECS to use task definition parameters to route logs to an AWS service or AWS Partner Network (APN) destination for log storage and analytics. 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

Consider the following when using FireLens for Amazon ECS:

- FireLens for Amazon ECS is supported for tasks that are hosted on both AWS Fargate on Linux and Amazon EC2. Windows containers that are on AWS Fargate don't support FireLens.
- FireLens for Amazon ECS is supported in AWS CloudFormation templates. For more information, see [AWS::ECS::TaskDefinition FirelensConfiguration](#) in the *AWS CloudFormation User Guide*
- FireLens listens on port 24224, so to ensure that the FireLens log router isn't reachable outside of the task, you must not allow inbound traffic on port 24224 in the security group your task uses. For tasks that use the `awsvpc` network mode, this is the security group associated with the task. For tasks using the `host` network mode, this is the security group that's associated with the Amazon EC2 instance

hosting the task. For tasks that use the bridge network mode, don't create any port mappings that use port 24224.

- For tasks that use the bridge network mode, the container with the FireLens configuration must start before any application containers that rely on it start. To control the start order of your containers, use dependency conditions in your task definition. For more information, see [Container dependency](#) (p. 116).

#### 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.

- The Amazon ECS-optimized Bottlerocket AMI doesn't support FireLens.
- By default, FireLens adds the cluster and task definition name and the Amazon Resource Name (ARN) of the cluster as metadata keys to your stdout/stderr container logs. The following is an example of the metadata format.

```
"ecs_cluster": "cluster-name",
"ecs_task_arn": "arn:aws:ecs:region:111122223333:task/cluster-
name/f2ad7dba413f45ddb4EXAMPLE",
"ecs_task_definition": "task-def-name:revision",
```

If you do not want the metadata in your logs, set `enable-ecs-log-metadata` to `false` in the `firelensConfiguration` section of the task definition.

```
"firelensConfiguration":{
  "type":"fluentbit",
  "options":{
    "enable-ecs-log-metadata":"false",
    "config-file-type":"file",
    "config-file-value":"/extra.conf"
  }
}
```

## 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, the task requires permission to call the `firehose:PutRecordBatch` API. For more information, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

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. 354).

- If your task is hosted on Fargate and you are pulling container images from Amazon ECR or referencing sensitive data from AWS Secrets Manager in your log configuration, then you must include the task execution IAM role.
- If you are specifying a custom configuration file that's 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 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"
      ]
    }
  ]
}
```

## Fluentd buffer limit

When you create a task definition, you can specify the number of events that are buffered in memory by specifying the value (in bytes) in the `log-driver-buffer-limit`. For more information, see [Fluentd logging driver](#) in the Docker documentation.

Use this option when there's high throughput, because Docker might run out of buffer memory and discard buffer messages so it can add new messages. The lost logs might make it difficult to troubleshoot. Setting the buffer limit might help to prevent this issue.

The following shows the syntax for specifying the `log-driver-buffer-limit`:

```
{
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:stable",
      "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",
        "log-driver-buffer-limit": "2097152"
      }
    },
    "dependsOn": [
      {
        "containerName": "log_router",
        "condition": "START"
      }
    ],
    "memoryReservation": 100
  }
]
```

Consider the following when using FireLens for Amazon ECS with the buffer limit option:

- This option is supported on the Amazon EC2 launch type and the Fargate launch type with platform version 1.4.0 or later.
- The option is only valid when `logDriver` is set to `awsfirelens`.
- The default buffer limit is 1 MiB.
- The valid values are 0 and 536870912 (512 MiB).
- The total amount of memory allocated at the task level must be greater than the amount of memory that's allocated for all the containers in addition to the memory buffer limit. The total amount of buffer memory specified must be less than 536870912 (512MiB) when you don't specify the container `memory` and `memoryReservation` values. More specifically, you can have an app container with the `awsfirelens` log driver and the `log-driver-buffer-limit` option set to 300 MiB. However, you won't be allowed to run tasks if you have more than two containers with the `log-driver-buffer-limit` set to 300 MiB ( $300 \text{ MiB} * 2 > 512 \text{ MiB}$ ).

## Using Fluent logger libraries or Log4j over TCP

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

`FLUENT_HOST`

The IP address that's assigned to the FireLens container.

## FLUENT\_PORT

The port that the Fluent Forward protocol is listening on.

You can use the `FLUENT_HOST` and `FLUENT_PORT` environment variables to log directly to the log router from code instead of going through `stdout`. For more information, see [fluent-logger-golang](#) on GitHub.

- [the section called “Using the AWS for Fluent Bit image” \(p. 148\)](#)
- [the section called “Creating a task definition that uses a FireLens configuration” \(p. 149\)](#)
- [the section called “Filtering logs using regular expressions” \(p. 152\)](#)
- [the section called “Example task definitions” \(p. 169\)](#)

## 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 Amazon ECR on both the Amazon ECR Public Gallery and in an Amazon ECR repository in most AWS Regions for high availability.

## Amazon ECR Public Gallery

The AWS for Fluent Bit image is available on the Amazon ECR Public Gallery. This is the recommended location to download the AWS for Fluent Bit image because it's a public repository and available to be used from all AWS Regions. For more information, see [aws-for-fluent-bit](#) on the Amazon ECR Public Gallery.

You can pull the AWS for Fluent Bit image from the Amazon ECR Public Gallery by specifying the repository URL with the desired image tag. The available image tags can be found on the **Image tags** tab on the Amazon ECR Public Gallery.

The following shows the syntax to use for the Docker CLI.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:tag
```

For example, you can pull the latest stable AWS for Fluent Bit image using this Docker CLI command.

```
docker pull public.ecr.aws/aws-observability/aws-for-fluent-bit:stable
```

### Note

Unauthenticated pulls are allowed, but have a lower rate limit than authenticated pulls. To authenticate using your AWS account before pulling, use the following command.

```
aws ecr-public get-login-password --region us-east-1 | docker login --username AWS  
--password-stdin public.ecr.aws
```

## Amazon ECR

The AWS for Fluent Bit image is available on Amazon ECR for high availability. These images are available in most AWS Regions, including AWS GovCloud (US).

The latest stable AWS for Fluent Bit image URI can be retrieved using the following command.

```
aws ssm get-parameters \
  --names /aws/service/aws-for-fluent-bit/stable \
  --region us-east-1
```

All versions of the AWS for Fluent Bit image can be listed using the following command to query the Systems Manager Parameter Store parameter.

```
aws ssm get-parameters-by-path \
  --path /aws/service/aws-for-fluent-bit \
  --region us-east-1
```

The latest stable AWS for Fluent Bit image can be referenced in an AWS CloudFormation template by referencing the Systems Manager parameter store name. The following is an example:

```
Parameters:
  FireLensImage:
    Description: Fluent Bit image for the FireLens Container
    Type: AWS::SSM::Parameter::Value<String>
    Default: /aws/service/aws-for-fluent-bit/stable
```

## 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 that contains a FireLens configuration. We recommend that the container be marked as `essential`.
- One or more application containers that contain a log configuration specifying the `awsfirelens` log driver.
- A task IAM role Amazon Resource Name (ARN) that contains the permissions needed for the task to route the logs.

When creating a new task definition using the AWS Management Console, there is a FireLens integration section that makes it easy to add a log router container. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

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

### Important

FireLens listens on port 24224. Therefore, to ensure that the FireLens log router isn't reachable outside of the task, you must not allow ingress traffic on port 24224 in the security group your task uses. For tasks that use the `awsvpc` network mode, this is the security group that's associated with the task. For tasks that use the `host` network mode, this is the security group that's associated with the Amazon EC2 instance hosting the task. For tasks that use the `bridge` network mode, don't create any port mappings that use port 24224.

The following task definition example defines a log router container that uses Fluent Bit to route its logs to CloudWatch Logs. It also defines an application container that uses a log configuration to route logs to Amazon Kinesis Data Firehose and sets the memory that's used to buffer events to the 2 MiB.

```
{
  "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:stable",
      "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",
          "log-driver-buffer-limit": "2097152"
        }
      },
      "memoryReservation": 100
    }
  ]
}
```

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

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

**Note**

FireLens manages the `match` configuration. This configuration isn't specified in your task definition.

## Using Amazon 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 Amazon Resource Name (ARN) of the task that the container is part of.
- `ecs_task_definition` – The task definition name and revision that the task is using.

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:stable",
      "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 that you're using. For more information, see [Fluentd Config File Syntax](#) and [Fluent Bit Configuration File](#).

In your custom configuration file, for tasks using the `bridge` or `awsvpc` network mode, don't set a `Fluentd` or `Fluent Bit` forward input over TCP because FireLens adds it to the input configuration.

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

`config-file-type`

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

### Note

Tasks that are hosted on AWS Fargate only support the `file` configuration file type.

`config-file-value`

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

### Important

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

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

### Important

To specify a custom configuration file that's hosted in Amazon S3, ensure you have created a task execution IAM role with the proper permissions. For more information, see [Required IAM permissions \(p. 145\)](#).



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

```
{
  "containerDefinitions":[
    {
      "essential":true,
      "image":"906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:stable",
      "name":"log_router",
      "firelensConfiguration":{
        "type":"fluentbit",
        "options":{
          "config-file-type":"s3 | file",
          "config-file-value":"arn:aws:s3:::mybucket/fluent.conf | filepath"
        }
      }
    }
  ]
}
```

**Note**

Tasks hosted on AWS Fargate only support the `file` configuration file type.

## 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].*$",
          "include-pattern":"^[aeiou]$"
        }
      }
    }
  ]
}
```

## Concatenate multiline or stack-trace log messages

Beginning with AWS for Fluent Bit version 2.22.0, a multiline filter is included. The multiline filter helps concatenate log messages that originally belong to one context but were split across multiple records or log lines. For more information on the multiline filter, see the [Fluent Bit documentation](#).

Common examples of split log messages are:

- Stack traces. Follow the steps below to concatenate messages split by newlines.
- Applications that print logs on multiple lines. Follow the steps below to concatenate messages split by newlines.
- Log messages that were split because they were longer than the specified runtime max buffer size. You can concatenate log messages split by the container runtime by following the example on GitHub: [FireLens Example: Concatenate Partial/Split Container Logs](#).

Compare the images of the logs below to determine if you need the multiline filter.

Before

The following image shows the CloudWatch Logs console with the default log setting with multiple entries for each line in a stack trace.

```
▼ 2022-04-28T15:47:56.595-05:00 {"container_id":"82ba37cada1d44d389b03e78ca574faa-527074092"}
{
  "container_id": "82ba37cada1d44d389b03e78ca574faa-527074092",
  "container_name": "app",
  "source": "stdout",
  "log": "    at com.myproject.module.MyProject.badMethod(MyProject.java:10)",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:123456789012:task/default/82ba37cada1d44d389b03e78ca574faa-527074092",
  "ecs_task_definition": "firelens-example-multiline:3"
}

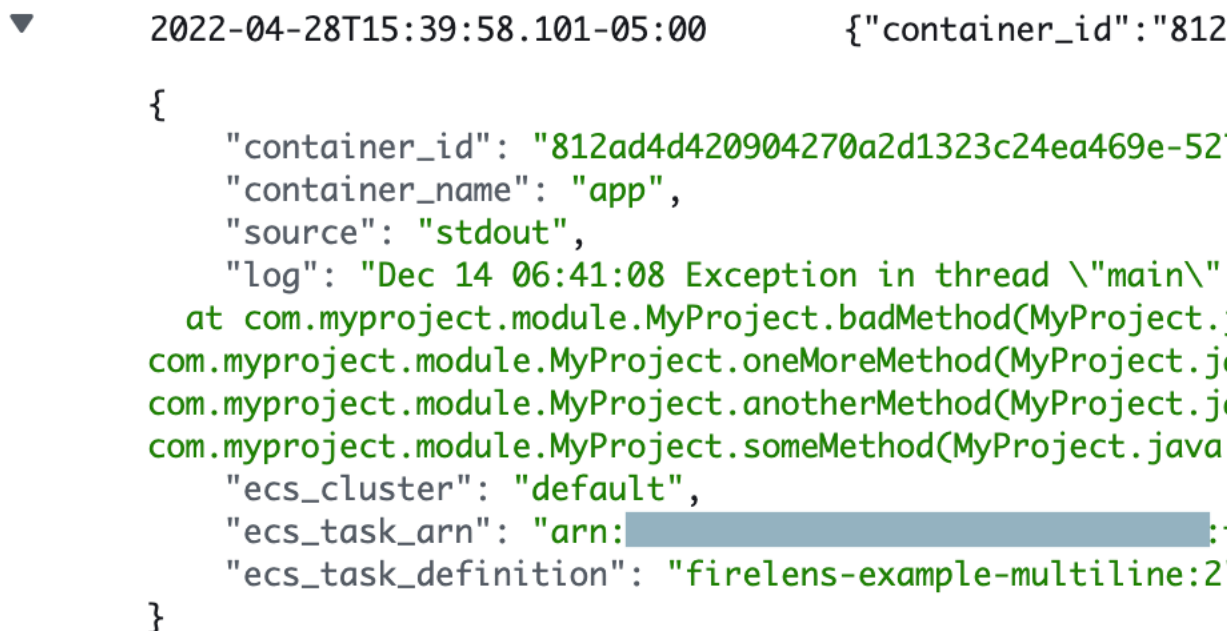
▼ 2022-04-28T15:47:56.595-05:00 {"log": "    at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:11)"}
{
  "log": "    at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:11)",
  "container_id": "82ba37cada1d44d389b03e78ca574faa-527074092",
  "container_name": "app",
  "source": "stdout",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:123456789012:task/default/82ba37cada1d44d389b03e78ca574faa-527074092",
  "ecs_task_definition": "firelens-example-multiline:3"
}

▼ 2022-04-28T15:47:56.595-05:00 {"source": "stdout", "log": "    at com.myproject.module.MyProject.anotherMethod(MyProject.java:12)"}
{
  "source": "stdout",
  "log": "    at com.myproject.module.MyProject.anotherMethod(MyProject.java:12)",
  "container_id": "82ba37cada1d44d389b03e78ca574faa-527074092",
  "container_name": "app",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:123456789012:task/default/82ba37cada1d44d389b03e78ca574faa-527074092",
  "ecs_task_definition": "firelens-example-multiline:3"
}

▼ 2022-04-28T15:47:56.596-05:00 {"container_name": "app", "source": "stdout", "log": "    at com.myproject.module.MyProject.someMethod(MyProject.java:13)"}
{
  "container_name": "app",
  "source": "stdout",
  "log": "    at com.myproject.module.MyProject.someMethod(MyProject.java:13)",
  "container_id": "82ba37cada1d44d389b03e78ca574faa-527074092",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:123456789012:task/default/82ba37cada1d44d389b03e78ca574faa-527074092",
  "ecs_task_definition": "firelens-example-multiline:3"
}
```

#### After

The following image shows the CloudWatch Logs console with the multiline log setting with a single entry that includes all the lines in a stack trace in the log field.



The screenshot shows a log entry in the CloudWatch Logs console. The log is a single line of JSON, but it contains a multiline stack trace in the "log" field. The stack trace is displayed in green text, indicating it was parsed as a single message. The log entry includes fields for container\_id, container\_name, source, log, ecs\_cluster, ecs\_task\_arn, and ecs\_task\_definition.

```
▼ 2022-04-28T15:39:58.101-05:00 {"container_id": "812ad4d420904270a2d1323c24ea469e-52"
{
  "container_id": "812ad4d420904270a2d1323c24ea469e-52",
  "container_name": "app",
  "source": "stdout",
  "log": "Dec 14 06:41:08 Exception in thread \"main\"
    at com.myproject.module.MyProject.badMethod(MyProject.j
    com.myproject.module.MyProject.oneMoreMethod(MyProject.j
    com.myproject.module.MyProject.anotherMethod(MyProject.j
    com.myproject.module.MyProject.someMethod(MyProject.java
  "ecs_cluster": "default",
  "ecs_task_arn": "arn: [REDACTED]:",
  "ecs_task_definition": "firelens-example-multiline:2
}
```

To parse logs and concatenate lines that were split because of newlines, you can use either of these two options.

- Create your own parser file that contains the rules to parse and concatenate lines that belong to the same message.
- Use a Fluent Bit built-in parser. For a list of languages supported by the Fluent Bit built-in parsers, see [Fluent Bit documentation](#).

The following tutorial walks you through the steps for each use case. The steps show you how to concatenate multilines and send the logs to Amazon CloudWatch. You can specify a different destination for your logs.

### Required IAM permissions

For each use case you must first make sure you have the necessary IAM permissions for the container agent to pull the container images from Amazon ECR and for the container to route logs to CloudWatch Logs.

For these permissions, you must have the following roles:

- A task IAM role.
- An Amazon ECS task execution IAM role.

#### Create the task IAM role

This task role grants the FireLens log router container the permissions needed to route the logs to the destination. In this example we are routing the logs to CloudWatch Logs. To create this role, create a policy with the permissions to create a log stream, log group, and write log events. Then associate the policy to the role.

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. Choose **JSON** and paste the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogStream",
      "logs:CreateLogGroup",
      "logs:PutLogEvents"
    ],
    "Resource": "*"
  }]
}
```

4. Choose **Next: Tags** and add any tags to the policy to help you organize them. Then choose **Next: Review**.
5. On the **Review policy** page, for **Name** type a unique name for the policy. In this example, we will use `ecs-policy-for-firelens`. You may specify an optional description for the policy as well.
6. Choose **Create policy** to finish.
7. In the navigation pane, choose **Roles** and then choose **Create Roles**.
8. In the **Trusted entity type** section, choose **AWS service**.
9. For **Use case**, choose **Elastic Container Service**.
10. Choose **Elastic Container Service Task** and then **Next**.
11. Associate the role with the `ecs-policy-for-firelens` policy you created and choose **Next**.
12. Enter a unique name for the role. In this example, use: `ecs-task-role-for-firelens`.

### Verify that you have an Amazon ECS task execution IAM role

You must have a task execution role to grant the container agent permission to pull the container images from Amazon ECR.

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then search for `ecsTaskExecutionRole`.
3. If you do not see the `ecsTaskExecutionRole` role, you must create the role. For information on how to create the role, see [Amazon ECS task execution IAM role](#) in the *Amazon Elastic Container Service Developer Guide*.

### Example: Use a parser that you create

In this example, you will complete the following steps:

1. Build and upload the image for a Fluent Bit container.
2. Build and upload the image for a demo multiline application that runs, fails, and generates a multiline stack trace.
3. Create the task definition and run the task.
4. View the logs to verify that messages that span multiple lines appear concatenated.

## Build and upload the image for a Fluent Bit container

This image will include the parser file where you specify the regular expression and a configuration file that references the parser file.

1. Create a folder with the name `FluentBitDockerImage`.
2. Within the folder, create a parser file that contains the rules to parse the log and concatenate lines that belong in the same message.
  - a. Paste the following contents in the parser file:

```
[MULTILINE_PARSER]
  name          multiline-regex-test
  type          regex
  flush_timeout 1000
  #
  # Regex rules for multiline parsing
  # -----
  #
  # configuration hints:
  #
  # - first state always has the name: start_state
  # - every field in the rule must be inside double quotes
  #
  # rules | state name | regex pattern | next state
  # -----|-----|-----|-----
  rule    "start_state" "/(<Dec \d+ \d+:\d+:\d+)(.*)/" "cont"
  rule    "cont"        "/^\s+at.*/"           "cont"
```

As you customize your regex pattern, we recommend you use a regular expression editor to test the expression.

- b. Save the file as `parsers_multiline.conf`.
3. Within the `FluentBitDockerImage` folder, create a custom configuration file that references the parser file that you created in the previous step.

For more information on the custom configuration file, see [Specifying a custom configuration file](#) in the *Amazon Elastic Container Service Developer Guide*

- a. Paste the following contents in the file:

```
[SERVICE]
  flush          1
  log_level      info
  parsers_file   /parsers_multiline.conf

[FILTER]
  name          multiline
  match         *
  multiline.key_content log
  multiline.parser multiline-regex-test
```

### Note

You must use the absolute path of the parser.

- b. Save the file as `extra.conf`.
4. Within the `FluentBitDockerImage` folder, create the Dockerfile with the Fluent Bit image and the parser and configuration files that you created.
    - a. Paste the following contents in the file:

```
FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest

ADD parsers_multiline.conf /parsers_multiline.conf
ADD extra.conf /extra.conf
```

- b. Save the file as Dockerfile.

- 5. Using the Dockerfile, build a custom Fluent Bit image with the parser and custom configuration files included.

**Note**

You can place the parser file and configuration file anywhere in the Docker image except / fluent-bit/etc/fluent-bit.conf as this file path is used by FireLens.

- a. Build the image: `docker build -t fluent-bit-multiline-image .`

Where: `fluent-bit-multiline-image` is the name for the image in this example.

- b. Verify that the image was created correctly: `docker images --filter reference=fluent-bit-multiline-image`

If successful, the output shows the image and the latest tag.

- 6. Upload the custom Fluent Bit image to Amazon Elastic Container Registry.

- a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

Where: `fluent-bit-multiline-repo` is the name for the repository and `us-east-1` is the region in this example.

The output gives you the details of the new repository.

- b. Tag your image with the `repositoryUri` value from the previous output: `docker tag fluent-bit-multiline-image repositoryUri`

Example: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

- c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`

In the output, the repository name changes from `fluent-bit-multiline-repo` to the `repositoryUri`.

- d. Authenticate to Amazon ECR by running the `aws ecr get-login-password` command and specifying the registry ID you want to authenticate to: `aws ecr get-login-password | docker login --username AWS --password-stdin registryID.dkr.ecr.region.amazonaws.com`

Example: `aws ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

A successful login message appears.

- e. Push the image to Amazon ECR: `docker push registryID.dkr.ecr.region.amazonaws.com/repository name`

Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

## Build and upload the image for a demo multiline application

This image will include a Python script file that runs the application and a sample log file.

When you run the task, the application simulates runs, then fails and creates a stack trace.

1. Create a folder named `multiline-app`: `mkdir multiline-app`
2. Create a Python script file.
  - a. Within the `multiline-app` folder, create a file and name it `main.py`.
  - b. Paste the following contents in the file:

```
import os
import time
file1 = open('/test.log', 'r')
Lines = file1.readlines()

count = 0

for i in range(10):
    print("app running normally...")
    time.sleep(1)

# Strips the newline character
for line in Lines:
    count += 1
    print(line.rstrip())
print(count)
print("app terminated.")
```

- c. Save the `main.py` file.
3. Create a sample log file.
  - a. Within the `multiline-app` folder, create a file and name it `test.log`.
  - b. Paste the following contents in the file:

```
single line...
Dec 14 06:41:08 Exception in thread "main" java.lang.RuntimeException: Something
has gone wrong, aborting!
    at com.myproject.module.MyProject.badMethod(MyProject.java:22)
    at com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)
    at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)
    at com.myproject.module.MyProject.someMethod(MyProject.java:10)
    at com.myproject.module.MyProject.main(MyProject.java:6)
another line...
```

- c. Save the `test.log` file.
4. Within the `multiline-app` folder, create the Dockerfile.
  - a. Paste the following contents in the file:

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log

RUN yum upgrade -y && yum install -y python3

WORKDIR /usr/local/bin

COPY main.py .
```



```
CMD ["python3", "main.py"]
```

- b. Save the Dockerfile file.
5. Using the Dockerfile, build an image.

- a. Build the image: `docker build -t multiline-app-image .`

Where: `multiline-app-image` is the name for the image in this example.

- b. Verify that the image was created correctly: `docker images --filter reference=multiline-app-image`

If successful, the output shows the image and the latest tag.

6. Upload the image to Amazon Elastic Container Registry.

- a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`

Where: `multiline-app-repo` is the name for the repository and `us-east-1` is the region in this example.

The output gives you the details of the new repository. Note the `repositoryUri` value as you will need it in the next steps.

- b. Tag your image with the `repositoryUri` value from the previous output: `docker tag multiline-app-image repositoryUri`

Example: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

- c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`

In the output, the repository name changes from `multiline-app-repo` to the `repositoryUri` value.

- d. Push the image to Amazon ECR: `docker push aws_account_id.dkr.ecr.region.amazonaws.com/repository name`

Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

## Create the task definition and run the task

1. Create a task definition file with the file name `multiline-task-definition.json`.
2. Paste the following contents in the `multiline-task-definition.json` file:

```
{
  "family": "firelens-example-multiline",
  "taskRoleArn": "task role ARN",
  "executionRoleArn": "execution role ARN",
  "containerDefinitions": [
    {
      "essential": true,
      "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-image:latest",
      "name": "log_router",
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": {
          "config-file-type": "file",
          "config-file-value": "/extra.conf"
        }
      }
    }
  ]
}
```

```
        },
        "memoryReservation": 50
    },
    {
        "essential": true,
        "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-
image:latest",
        "name": "app",
        "logConfiguration": {
            "logDriver": "awsfirelens",
            "options": {
                "Name": "cloudwatch_logs",
                "region": "us-east-1",
                "log_group_name": "multiline-test/application",
                "auto_create_group": "true",
                "log_stream_prefix": "multiline-"
            }
        }
    },
    "memoryReservation": 100
}
],
"requiresCompatibilities": ["FARGATE"],
"networkMode": "awsvpc",
"cpu": "256",
"memory": "512"
}
```

Replace the following in the `multiline-task-definition.json` task definition:

a. *task role ARN*

To find the task role ARN, go to the IAM console. Choose **Roles** and find the `ecs-task-role-for-firelens` task role that you created. Choose the role and copy the **ARN** that appears in the **Summary** section.

b. *execution role ARN*

To find the execution role ARN, go to the IAM console. Choose **Roles** and find the `ecsTaskExecutionRole` role. Choose the role and copy the **ARN** that appears in the **Summary** section.

c. *aws\_account\_id*

To find your `aws_account_id`, log into the AWS Management Console. Choose your user name on the top right and copy your Account ID.

d. *us-east-1*

Replace the region if necessary.

3. Register the task definition file: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region us-east-1`
4. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
5. In the navigation pane, choose **Task Definitions** and then choose the `firelens-example-multiline` family because we registered the task definition to this family in the first line of the task definition above.
6. Choose the latest version.
7. Choose the **Actions, Run Task**.
8. For **Launch type**, choose **Fargate**.
9. For **Subnets**, choose the available subnets for your task.
10. Choose **Run Task**.

## Verify that multiline log messages in Amazon CloudWatch appear concatenated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. From the navigation pane, expand **Logs** and choose **Log groups**.
3. Choose the multiline-test/application log group.
4. Choose the log. View messages. Lines that matched the rules in the parser file are concatenated and appear as a single message.

The following log snippet shows lines concatenated in a single Java stack trace event:

```
{
  "container_id": "xxxxxx",
  "container_name": "app",
  "source": "stdout",
  "log": "Dec 14 06:41:08 Exception in thread \"main\"
java.lang.RuntimeException: Something has gone wrong, aborting!\n
at com.myproject.module.MyProject.badMethod(MyProject.java:22)\n    at
com.myproject.module.MyProject.oneMoreMethod(MyProject.java:18)\n
at com.myproject.module.MyProject.anotherMethod(MyProject.java:14)\n
at com.myproject.module.MyProject.someMethod(MyProject.java:10)\n    at
com.myproject.module.MyProject.main(MyProject.java:6)",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
  "ecs_task_definition": "firelens-example-multiline:2"
}
```

The following log snippet shows how the same message appears with just a single line if you run an ECS container that is not configured to concatenate multiline log messages.

```
{
  "log": "Dec 14 06:41:08 Exception in thread \"main\" java.lang.RuntimeException:
Something has gone wrong, aborting!",
  "container_id": "xxxxxx-xxxxxx",
  "container_name": "app",
  "source": "stdout",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
  "ecs_task_definition": "firelens-example-multiline:3"
}
```

## Example: Use a Fluent Bit built-in parser

In this example, you will complete the following steps:

1. Build and upload the image for a Fluent Bit container.
2. Build and upload the image for a demo multiline application that runs, fails, and generates a multiline stack trace.
3. Create the task definition and run the task.
4. View the logs to verify that messages that span multiple lines appear concatenated.

### Build and upload the image for a Fluent Bit container

This image will include a configuration file that references the Fluent Bit parser.

1. Create a folder with the name `FluentBitDockerImage`.

2. Within the `FluentBitDockerImage` folder, create a custom configuration file that references the Fluent Bit built-in parser file.

For more information on the custom configuration file, see [Specifying a custom configuration file](#) in the *Amazon Elastic Container Service Developer Guide*

- a. Paste the following contents in the file:

```
[FILTER]
  name          multiline
  match         *
  multiline.key_content log
  multiline.parser go
```

- b. Save the file as `extra.conf`.

3. Within the `FluentBitDockerImage` folder, create the Dockerfile with the Fluent Bit image and the parser and configuration files that you created.

- a. Paste the following contents in the file:

```
FROM public.ecr.aws/aws-observability/aws-for-fluent-bit:latest
ADD extra.conf /extra.conf
```

- b. Save the file as `Dockerfile`.

4. Using the Dockerfile, build a custom Fluent Bit image with the custom configuration file included.

**Note**

You can place the configuration file anywhere in the Docker image except `/fluent-bit/etc/fluent-bit.conf` as this file path is used by FireLens.

- a. Build the image: `docker build -t fluent-bit-multiline-image .`

Where: `fluent-bit-multiline-image` is the name for the image in this example.

- b. Verify that the image was created correctly: `docker images --filter reference=fluent-bit-multiline-image`

If successful, the output shows the image and the latest tag.

5. Upload the custom Fluent Bit image to Amazon Elastic Container Registry.

- a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name fluent-bit-multiline-repo --region us-east-1`

Where: `fluent-bit-multiline-repo` is the name for the repository and `us-east-1` is the region in this example.

The output gives you the details of the new repository.

- b. Tag your image with the `repositoryUri` value from the previous output: `docker tag fluent-bit-multiline-image repositoryUri`

Example: `docker tag fluent-bit-multiline-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-repo`

- c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`

In the output, the repository name changes from `fluent-bit-multiline-repo` to the `repositoryUri`.

- d. Authenticate to Amazon ECR by running the `aws ecr get-login-password` command and specifying the registry ID you want to authenticate to: `aws ecr get-login-`

```
password | docker login --username AWS --password-stdin registry  
ID.dkr.ecr.region.amazonaws.com
```

Example: `ecr get-login-password | docker login --username AWS --password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com`

A successful login message appears.

- e. Push the image to Amazon ECR: `docker push registry  
ID.dkr.ecr.region.amazonaws.com/repository name`

Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/fluently-bit-multiline-repo`

## Build and upload the image for a demo multiline application

This image will include a Python script file that runs the application and a sample log file.

1. Create a folder named `multiline-app`: `mkdir multiline-app`
2. Create a Python script file.
  - a. Within the `multiline-app` folder, create a file and name it `main.py`.
  - b. Paste the following contents in the file:

```
import os  
import time  
file1 = open('/test.log', 'r')  
Lines = file1.readlines()  
  
count = 0  
  
for i in range(10):  
    print("app running normally...")  
    time.sleep(1)  
  
# Strips the newline character  
for line in Lines:  
    count += 1  
    print(line.rstrip())  
print(count)  
print("app terminated.")
```

- c. Save the `main.py` file.
3. Create a sample log file.
  - a. Within the `multiline-app` folder, create a file and name it `test.log`.
  - b. Paste the following contents in the file:

```
panic: my panic  
  
goroutine 4 [running]:  
panic(0x45cb40, 0x47ad70)  
  /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8 sp=0xc42003f710  
  pc=0x422f7c  
main.main.func1(0xc420024120)  
  foo.go:6 +0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339  
runtime.goexit()  
  /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0 sp=0xc42003f7d8  
  pc=0x44b4d1  
created by main.main
```

```
foo.go:5 +0x58

goroutine 1 [chan receive]:
runtime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)
    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00
pc=0x42503c
runtime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)
    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30
pc=0x42512e
runtime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)
    /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70
pc=0x4046b4
runtime.chanrecv1(0xc420024120, 0x0)
    /usr/local/go/src/runtime/chan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20
pc=0x40439b
main.main()
    foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef
runtime.main()
    /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80
pc=0x424bad
runtime.goexit()
    /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc420053fe8 sp=0xc420053fe0
pc=0x44b4d1

goroutine 2 [force gc (idle)]:
runtime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)
    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738
pc=0x42503c
runtime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)
    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768
pc=0x42512e
runtime.forcegchelper()
    /usr/local/go/src/runtime/proc.go:238 +0xcc fp=0xc42003e7e0 sp=0xc42003e7a8
pc=0x424e5c
runtime.goexit()
    /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8 sp=0xc42003e7e0
pc=0x44b4d1
created by runtime.init.4
    /usr/local/go/src/runtime/proc.go:227 +0x35

goroutine 3 [GC sweep wait]:
runtime.gopark(0x4739b8, 0x4ad7e0, 0x46fdd2, 0xd, 0x419914, 0x1)
    /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003ef60 sp=0xc42003ef30
pc=0x42503c
runtime.goparkunlock(0x4ad7e0, 0x46fdd2, 0xd, 0x14, 0x1)
    /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003efa0 sp=0xc42003ef60
pc=0x42512e
runtime.bgsweep(0xc42001e150)
    /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3 fp=0xc42003efd8 sp=0xc42003efa0
pc=0x419973
runtime.goexit()
    /usr/local/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003efe0 sp=0xc42003efd8
pc=0x44b4d1
created by runtime.gcenable
    /usr/local/go/src/runtime/mgc.go:216 +0x58
one more line, no multiline
```

- c. Save the test.log file.
4. Within the multiline-app folder, create the Dockerfile.
  - a. Paste the following contents in the file:

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
ADD test.log /test.log
```

```
RUN yum upgrade -y && yum install -y python3  
WORKDIR /usr/local/bin  
COPY main.py .  
CMD ["python3", "main.py"]
```

- b. Save the Dockerfile file.
5. Using the Dockerfile, build an image.
  - a. Build the image: `docker build -t multiline-app-image .`  
  
Where: `multiline-app-image` is the name for the image in this example.
  - b. Verify that the image was created correctly: `docker images --filter reference=multiline-app-image`  
  
If successful, the output shows the image and the latest tag.
6. Upload the image to Amazon Elastic Container Registry.
  - a. Create an Amazon ECR repository to store the image: `aws ecr create-repository --repository-name multiline-app-repo --region us-east-1`  
  
Where: `multiline-app-repo` is the name for the repository and `us-east-1` is the region in this example.  
  
The output gives you the details of the new repository. Note the `repositoryUri` value as you will need it in the next steps.
  - b. Tag your image with the `repositoryUri` value from the previous output: `docker tag multiline-app-image repositoryUri`  
  
Example: `docker tag multiline-app-image xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`
  - c. Run the docker image to verify it ran correctly: `docker images --filter reference=repositoryUri`  
  
In the output, the repository name changes from `multiline-app-repo` to the `repositoryUri` value.
  - d. Push the image to Amazon ECR: `docker push aws_account_id.dkr.ecr.region.amazonaws.com/repository name`  
  
Example: `docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/multiline-app-repo`

### Create the task definition and run the task

1. Create a task definition file with the file name `multiline-task-definition.json`.
2. Paste the following contents in the `multiline-task-definition.json` file:

```
{  
  "family": "firelens-example-multiline",  
  "taskRoleArn": "task role ARN",  
  "executionRoleArn": "execution role ARN",  
  "containerDefinitions": [  
    {  
      "essential": true,
```

```
        "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/fluent-bit-multiline-image:latest",
        "name": "log_router",
        "firelensConfiguration": {
            "type": "fluentbit",
            "options": {
                "config-file-type": "file",
                "config-file-value": "/extra.conf"
            }
        },
        "memoryReservation": 50
    },
    {
        "essential": true,
        "image": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/multiline-app-image:latest",
        "name": "app",
        "logConfiguration": {
            "logDriver": "awsfirelens",
            "options": {
                "Name": "cloudwatch_logs",
                "region": "us-east-1",
                "log_group_name": "multiline-test/application",
                "auto_create_group": "true",
                "log_stream_prefix": "multiline-"
            }
        },
        "memoryReservation": 100
    }
],
"requiresCompatibilities": ["FARGATE"],
"networkMode": "awsvpc",
"cpu": "256",
"memory": "512"
}
```

Replace the following in the `multiline-task-definition.json` task definition:

a. *task role ARN*

To find the task role ARN, go to the IAM console. Choose **Roles** and find the `ecs-task-role-for-firelens` task role that you created. Choose the role and copy the **ARN** that appears in the **Summary** section.

b. *execution role ARN*

To find the execution role ARN, go to the IAM console. Choose **Roles** and find the `ecsTaskExecutionRole` role. Choose the role and copy the **ARN** that appears in the **Summary** section.

c. *aws\_account\_id*

To find your `aws_account_id`, log into the AWS Management Console. Choose your user name on the top right and copy your Account ID.

d. *us-east-1*

Replace the region if necessary.

3. Register the task definition file: `aws ecs register-task-definition --cli-input-json file://multiline-task-definition.json --region us-east-1`
4. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.



5. In the navigation pane, choose **Task Definitions** and then choose the `firelens-example-multiline` family because we registered the task definition to this family in the first line of the task definition above.
6. Choose the latest version.
7. Choose the **Actions, Run Task**.
8. For **Launch type**, choose **Fargate**.
9. For **Subnets**, choose the available subnets for your task.
10. Choose **Run Task**.

### Verify that multiline log messages in Amazon CloudWatch appear concatenated

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. From the navigation pane, expand **Logs** and choose **Log groups**.
3. Choose the `multiline-test/applicatio` log group.
4. Choose the log and view the messages. Lines that matched the rules in the parser file are concatenated and appear as a single message.

The following log snippet shows a Go stack trace that is concatenated into a single event:

```
{
  "log": "panic: my panic\n\nngoroutine 4 [running]:\npanic(0x45cb40,\n0x47ad70)\n /usr/local/go/src/runtime/panic.go:542 +0x46c fp=0xc42003f7b8\nsp=0xc42003f710 pc=0x422f7c\nmain.main.func1(0xc420024120)\n foo.go:6\n+0x39 fp=0xc42003f7d8 sp=0xc42003f7b8 pc=0x451339\nruntime.goexit()\n /usr/\nlocal/go/src/runtime/asm_amd64.s:2337 +0x1 fp=0xc42003f7e0 sp=0xc42003f7d8\npc=0x44b4d1\ncreated by main.main\n foo.go:5 +0x58\n\nngoroutine 1 [chan receive]:\n\nruntime.gopark(0x4739b8, 0xc420024178, 0x46fcd7, 0xc, 0xc420028e17, 0x3)\n /usr/\nlocal/go/src/runtime/proc.go:280 +0x12c fp=0xc420053e30 sp=0xc420053e00 pc=0x42503c\n\nruntime.goparkunlock(0xc420024178, 0x46fcd7, 0xc, 0x1000f010040c217, 0x3)\n\n /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc420053e70 sp=0xc420053e30\npc=0x42512e\nruntime.chanrecv(0xc420024120, 0x0, 0xc420053f01, 0x4512d8)\n\n /usr/local/go/src/runtime/chan.go:506 +0x304 fp=0xc420053f20 sp=0xc420053e70\npc=0x4046b4\nruntime.chanrecv1(0xc420024120, 0x0)\n /usr/local/go/src/runtime/\nchan.go:388 +0x2b fp=0xc420053f50 sp=0xc420053f20 pc=0x40439b\nmain.main()\n foo.go:9 +0x6f fp=0xc420053f80 sp=0xc420053f50 pc=0x4512ef\nruntime.main()\n\n /usr/local/go/src/runtime/proc.go:185 +0x20d fp=0xc420053fe0 sp=0xc420053f80\npc=0x424bad\nruntime.goexit()\n /usr/local/go/src/runtime/asm_amd64.s:2337\n+0x1 fp=0xc420053fe0 sp=0xc420053fe0 pc=0x44b4d1\n\nngoroutine 2 [force gc\n(idle)]:\n\nruntime.gopark(0x4739b8, 0x4ad720, 0x47001e, 0xf, 0x14, 0x1)\n /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003e768 sp=0xc42003e738\npc=0x42503c\nruntime.goparkunlock(0x4ad720, 0x47001e, 0xf, 0xc420000114, 0x1)\n\n /usr/local/go/src/runtime/proc.go:286 +0x5e fp=0xc42003e7a8 sp=0xc42003e768\npc=0x42512e\nruntime.forcegchelper()\n /usr/local/go/src/runtime/proc.go:238 +0xcc\nfp=0xc42003e7e0 sp=0xc42003e7a8 pc=0x424e5c\nruntime.goexit()\n /usr/local/go/src/\nruntime/asm_amd64.s:2337 +0x1 fp=0xc42003e7e8 sp=0xc42003e7e0 pc=0x44b4d1\n\ncreated\nby runtime.init.4\n /usr/local/go/src/runtime/proc.go:227 +0x35\n\nngoroutine 3 [GC\nsweep wait]:\n\nruntime.gopark(0x4739b8, 0x4ad7e0, 0x46fdd2, 0xd, 0x419914, 0x1)\n\n /usr/local/go/src/runtime/proc.go:280 +0x12c fp=0xc42003ef60 sp=0xc42003ef30\npc=0x42503c\nruntime.goparkunlock(0x4ad7e0, 0x46fdd2, 0xd, 0x14, 0x1)\n\n /usr/\nlocal/go/src/runtime/proc.go:286 +0x5e fp=0xc42003efa0 sp=0xc42003ef60 pc=0x42512e\n\nruntime.bgsweep(0xc42001e150)\n\n /usr/local/go/src/runtime/mgcsweep.go:52 +0xa3\nfp=0xc42003efd8 sp=0xc42003efa0 pc=0x419973\nruntime.goexit()\n /usr/local/go/src/\nruntime/asm_amd64.s:2337 +0x1 fp=0xc42003efe0 sp=0xc42003efd8 pc=0x44b4d1\n\ncreated by\nruntime.gcenable\n /usr/local/go/src/runtime/mgc.go:216 +0x58",
  "container_id": "xxxxxx-xxxxxx",
  "container_name": "app",
  "source": "stdout",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
```

```
    "ecs_task_definition": "firelens-example-multiline:2"
  }
```

The following log snippet shows how the same event appears if you run an ECS container that is not configured to concatenate multiline log messages. The log field contains a single line.

```
{
  "log": "panic: my panic",
  "container_id": "xxxxxx-xxxxxx",
  "container_name": "app",
  "source": "stdout",
  "ecs_cluster": "default",
  "ecs_task_arn": "arn:aws:ecs:us-east-1:xxxxxxxxxxxx:task/default/xxxxxx",
  "ecs_task_definition": "firelens-example-multiline:3"
}
```

### Note

If your logs go to log files instead of the standard output, we recommend specifying the `multiline.parser` and `multiline.key_content` configuration parameters in the [Tail input plugin](#) instead of the Filter.

## Example task definitions

The following are some example task definitions demonstrating common custom log routing options. For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

### Topics

- [Forwarding logs to CloudWatch Logs \(p. 169\)](#)
- [Forwarding logs to an Amazon Kinesis Data Firehose delivery stream \(p. 170\)](#)
- [Forwarding logs to an Amazon OpenSearch Service domain \(p. 171\)](#)
- [Parsing container logs that are serialized JSON \(p. 172\)](#)
- [Forwarding to an external Fluentd or Fluent Bit \(p. 173\)](#)

## Forwarding logs to CloudWatch Logs

### Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

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 AWS Region it exists in. To have Fluent Bit create the log group on your behalf, specify `"auto_create_group": "true"`, to set the `fluentd-buffer-limit` use `log-driver-buffer-limit`. You can also specify the task ID as the 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": "httpd",
        "name": "app",
        "logConfiguration": {
            "logDriver": "awsfirelens",
            "options": {
                "Name": "cloudwatch",
                "region": "us-west-2",
                "log_group_name": "firelens-blog",
                "auto_create_group": "true",
                "log_stream_prefix": "from-fluent-bit",
                "log-driver-buffer-limit": "2097152"
            }
        },
        "memoryReservation": 100
    }
]
}

```

## Forwarding logs to an Amazon Kinesis Data Firehose delivery stream

### Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

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:stable",
            "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 logs to an Amazon OpenSearch Service domain

### Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

The following task definition example demonstrates how to specify a log configuration that forwards logs to an Amazon OpenSearch Service domain. The Amazon OpenSearch Service domain must already exist. For more information, see [What is Amazon OpenSearch Service](#) in the *Amazon OpenSearch Service Developer Guide*.

In the log configuration options, specify the log options required for OpenSearch Service integration. For more information, see [Fluent Bit for Amazon OpenSearch Service](#).

```

{
  "family": "firelens-example-opensearch",
  "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:stable",
      "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": "es",
                "Host": "vpc-fake-domain-ke7thhzo07jawrhmz6mb7ite7y.us-  
west-2.es.amazonaws.com",
                "Port": "443",
                "Index": "my_index",
                "Type": "my_type",
                "AWS_Auth": "On",
                "AWS_Region": "us-west-2",
                "tls": "On"
            }
        },
        "memoryReservation": 100
    }
}
```

## Parsing container logs that are serialized JSON

### Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

Beginning with AWS for Fluent Bit version 1.3, there's a JSON parser that's included in the AWS for Fluent Bit image. The following example shows how to reference the JSON parser in the FireLens configuration of your task definition.

```
"firelensConfiguration": {
    "type": "fluentbit",
    "options": {
        "config-file-type": "file",
        "config-file-value": "/fluent-bit/configs/parse-json.conf"
    }
},
```

The Fluent Bit config file parses any logs that are in JSON (for example, if the logs at your destination looked like the following without JSON parsing).

```
{
    "source": "stdout",
    "log": "{\"requestID\": \"b5d716fca19a4252ad90e7b8ec7cc8d2\", \"requestInfo\":  
{\"ipAddress\": \"204.16.5.19\", \"path\": \"/activate\", \"user\": \"TheDoctor\"}}\",  
    \"container_id\": \"e54cccfac2b87417f71877907f67879068420042828067ae0867e60a63529d35\",  
    \"container_name\": \"/ecs-demo-6-container2-a4eafbb3d4c7f1e16e00\"  
    \"ecs_cluster\": \"mycluster\",  
    \"ecs_task_arn\": \"arn:aws:ecs:us-east-2:01234567891011:task/  
mycluster/3de392df-6bfa-470b-97ed-aa6f482cd7a6\",  
    \"ecs_task_definition\": \"demo:7\"  
    \"ec2_instance_id\": \"i-06bc83dbc2ac2fd8\"  
}
```

With the JSON parsing, the log looks like the following.

```
{
    "source": "stdout",
    "container_id": "e54cccfac2b87417f71877907f67879068420042828067ae0867e60a63529d35",
```

```

    "container_name": "/ecs-demo-6-container2-a4eafbb3d4c7f1e16e00"
    "ecs_cluster": "mycluster",
    "ecs_task_arn": "arn:aws:ecs:us-east-2:01234567891011:task/
mycluster/3de392df-6bfa-470b-97ed-aa6f482cd7a6",
    "ecs_task_definition": "demo:7"
    "ec2_instance_id": "i-06bc83dbc2ac2fdf8"
    "requestID": "b5d716fca19a4252ad90e7b8ec7cc8d2",
    "requestInfo": {
        "ipAddress": "204.16.5.19",
        "path": "/activate",
        "user": "TheDoctor"
    }
}

```

The serialized JSON is expanded into top level fields in the final JSON output. For more information about JSON parsing, see [Parser](#) in the Fluent Bit documentation.

## Forwarding to an external Fluentd or Fluent Bit

### Note

For more examples, see [Amazon ECS FireLens examples](#) on GitHub.

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:stable",
      "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 task definition. This provides a way to reference container images that exist in private registries outside of AWS that require authentication in your task definitions. This feature is supported by tasks hosted on Fargate, Amazon EC2 instances, and external instances using Amazon ECS Anywhere.

### Important

If your task definition references an image that's stored in Amazon ECR, this topic doesn't apply. For more information, see [Using Amazon ECR Images with Amazon ECS](#) in the *Amazon Elastic Container Registry User Guide*.

For tasks hosted on Fargate, this feature requires platform version 1.2.0 or later. For information, see [AWS Fargate platform versions \(p. 58\)](#).

Within your container definition, specify the `repositoryCredentials` object with the details of the secret that you created. The secret you reference can be from a different AWS Region or a different account than the task using it.

### Note

When using the Amazon ECS API, AWS CLI, or AWS SDK, if the secret exists in the same AWS Region as the task that you're launching then you can use either the full ARN or name of the secret. If the secret exists in a different account, the full ARN of the secret must be specified. When using the AWS Management Console, the full ARN of the secret must be specified always.

The following is a snippet of a task definition that shows the required parameters:

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

## 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 \(p. 354\)](#).

To provide access to the secrets that you create, 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 Amazon Resource Name (ARN) for your custom key must be added as a resource.

The following is an example inline policy that adds the permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
"Action": [
    "kms:Decrypt",
    "ssm:GetParameters",
    "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**, enter 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 instructions on 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 that 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**.
5. For **task definition Name**, enter 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**. This role authorizes Amazon ECS to pull private images for your task. For more information, see [Required IAM permissions for private registry authentication \(p. 174\)](#).



### Important

If the **Task execution role** field doesn't appear, choose **Configure via JSON** and 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 \(p. 97\)](#).
  - g. Choose **Add**.
8. When your containers are added, choose **Create**.

## Specifying sensitive data

You can use Amazon ECS 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 \(p. 176\)](#)
- [Specifying sensitive data using Systems Manager Parameter Store \(p. 182\)](#)

## 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 the full contents of a secret, a specific JSON key within a secret, or a specific version of a secret to inject. This 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*.

## Considerations for specifying sensitive data using Secrets Manager

Consider the following when using Secrets Manager to specify sensitive data for containers.

- For Amazon ECS tasks on AWS Fargate, consider the following:
  - To inject the full content of a secret as an environment variable or in a log configuration, you must use platform version 1.3.0 or later. For information, see [AWS Fargate platform versions \(p. 58\)](#).
  - To inject a specific JSON key or version of a secret as an environment variable or in a log configuration, you must use platform version 1.4.0 or later (Linux) or 1.0.0 (Windows). For information, see [AWS Fargate platform versions \(p. 58\)](#).
- Only secrets that store text data, which are secrets created with the `SecretString` parameter of the `CreateSecret` API, are supported. Secrets that store binary data, which are secrets created with the `SecretBinary` parameter of the `CreateSecret` API aren't supported.
- When using a task definition that references Secrets Manager secrets to retrieve sensitive data for your containers, if you're 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 doesn't receive the updated value automatically. You must either launch a new task or if your task is part of a service that you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- The VPC your task uses must have DNS resolution enabled.

## 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 \(p. 354\)](#).

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're 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 must 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
```

The following section describes the additional parameters. These parameters are optional, but if you don't 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 don't 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 can't 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're 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 can't 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're 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 how to create a Secrets Manager secret and inject it into a container as an environment variable, see [Tutorial: Specifying sensitive data using Secrets Manager secrets](#) (p. 408).

## Example container definitions

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

### Example referencing a full secret

The following is a snippet of a task definition that shows 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 that's associated with it.

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

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

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

### Example referencing a specific secret version

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

```
{
  "ARN": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-AbCdEf",
  "Name": "appauthexample",
  "Description": "Example of a secret containing application authorization data.",
  "RotationEnabled": false,
  "LastChangedDate": 1581968848.926,
  "LastAccessedDate": 1581897600.0,
  "Tags": [],
  "VersionIdsToStages": {
    "871d9eca-18aa-46a9-8785-981ddEXAMPLE": [
      "AWSCURRENT"
    ],
    "9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE": [
      "AWSPREVIOUS"
    ]
  }
}
```

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

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

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

```
{
  "containerDefinitions": [{
    "secrets": [{
      "name": "environment_variable_name",
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
    }]
  }]
}
```

## 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 Amazon Resource Name (ARN) of the Secrets Manager secret that contains the sensitive data to present to the container.

The following is a snippet of a task definition showing the format when referencing a 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 a database name, server address, and TCP port, and so on. You can add as many pairs as you need to store the information that 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 KMS key specifically for this secret. To create your own KMS key, you must have permissions to create KMS keys 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 that you entered as a new secret in Secrets Manager.

## Creating a task definition in the classic console that references sensitive data

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

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**.
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 \(p. 174\)](#).

#### Important

If the **Task execution role** field doesn't 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 that's 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 \(p. 97\)](#).
  - h. Choose **Add**.
7. When your containers are added, choose **Create**.

## Specifying sensitive data using Systems Manager Parameter Store

You can use Amazon ECS 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 \(p. 182\)](#)
- [Required IAM permissions for Amazon ECS secrets \(p. 183\)](#)
- [Injecting sensitive data as an environment variable \(p. 183\)](#)
- [Injecting sensitive data in a log configuration \(p. 184\)](#)
- [Creating an AWS Systems Manager Parameter Store parameter \(p. 184\)](#)
- [Creating a Task Definition in the classic console that references sensitive data \(p. 185\)](#)

## Considerations for specifying sensitive data using Systems Manager Parameter Store

Consider the following when specifying sensitive data for containers using Systems Manager Parameter Store parameters.

- The Systems Manager Parameter Store parameter must exist in the same account that the tasks are run in.

- For tasks hosted on Fargate, this feature requires that your task use platform version 1.3.0 or later (for Linux) or 1.0.0 or later (for Windows). For information, see [AWS Fargate platform versions \(p. 58\)](#).
- 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 doesn't 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.

## 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 \(p. 354\)](#).

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're referencing a Systems Manager Parameter Store parameter in a task definition.
- `secretsmanager:GetSecretValue`—Required if you're 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 must 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 Amazon Resource Name (ARN) of the Systems Manager Parameter Store parameter that contains the sensitive data to present to the container.

The following is a snippet of a task definition that shows the format that's used when referencing a Systems Manager Parameter Store parameter. If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching, then you can use either the full ARN or name of the parameter. If the parameter exists in a different Region, then 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 Amazon Resource Name (ARN) of the Systems Manager Parameter Store parameter that contains the sensitive data to present to the container.

### Important

If the Systems Manager Parameter Store parameter exists in the same AWS 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 a Systems Manager Parameter Store parameter.

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

## 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** and then **Create parameter**.
3. For **Name**, enter a hierarchy and a parameter name (for example, `/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 key ID, a KMS key ARN, an alias name, or an alias ARN, then the system uses `alias/aws/`

`ssm`. This is the default KMS key 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 key 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 in the classic console 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**.
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**. The role that's created authorizes Amazon ECS to pull private images for your task. For more information, see [Required IAM permissions for private registry authentication](#) (p. 174).

#### Important

If the **Task execution role** field doesn't appear, choose **Configure via JSON** and 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 that's in the `secrets` section of a container definition.
- ii. For **Value**, choose **ValueFrom**. For **Add value**, enter the full Amazon Resource Name (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 AWS Region as the task you are launching, you can use either the full ARN or name of the secret. If the parameter exists in a different Region, then the full ARN must be specified.

- 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 AWS Region as the task you are launching, 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 \(p. 97\)](#).
  - h. Choose **Add**.
7. When your containers are added, choose **Create**.

## Specifying environment variables

**Important**

We recommend storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters. For more information, see [Specifying sensitive data \(p. 176\)](#).

Environment variables specified in the task definition are readable by all IAM users and roles that are allowed the `DescribeTaskDefinition` action for the task definition.

Environment variable files are objects in Amazon S3 and all Amazon S3 security considerations apply. See the below section [the section called "Required IAM permissions" \(p. 188\)](#).

You can pass environment variables to your containers in the following ways:

- Individually using the `environment` container definition parameter. This maps to the `--env` option to [docker run](#).
- In bulk, using the `environmentFiles` container definition parameter to list one or more files that contain the environment variables. The file must be hosted in Amazon S3. This maps to the `--env-file` option to [docker run](#).

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

We don't enforce a size limit on the environment variables, but a large environment variables file might fill up the disk space. Each task that uses an environment variables file causes a copy of the file to be downloaded to disk. We remove the file as part of the task cleanup.

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

Consider the following when specifying an environment variable file in a container definition.

- For Amazon ECS tasks on AWS Fargate, your tasks must use platform version 1.4.0 or later (Linux) to use this feature. For more information, see [AWS Fargate platform versions](#) (p. 58).

Verify that the variable is supported for the operating system platform. For more information, see [the section called “Container definitions”](#) (p. 100) and [the section called “Other task definition parameters”](#) (p. 123).

- The file must use the .env file extension and UTF-8 encoding.

- Each line in an environment file must contain an environment variable in `VARIABLE=VALUE` format. Spaces or quotation marks are included as part of the values. Lines beginning with `#` are treated as comments and are ignored. For more information about the environment variable file syntax, see [Declare default environment variables in file](#).

The following is the appropriate syntax.

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

- If there are environment variables specified using the `environment` parameter in a container definition, they take precedence over the variables contained within an environment file.
- If multiple environment files are specified and they contain the same variable, they're processed in order of entry. This means that the first value of the variable is used and subsequent values of duplicate variables are ignored. We recommend that you use unique variable names.
- If an environment file is specified as a container override, it's used. Moreover, any other environment files that are specified in the container definition is ignored.

## 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 \(p. 354\)](#).

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`

In the following example, the permissions are added to the inline policy.

```
{
  "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 JSON task definition examples that you can use to start creating your own task definitions.

You can copy the examples, and then paste them when you use the **Configure via JSON** option in the classic consoles. Make sure to customize the examples, such as using your account ID. For more information, see [Creating a task definition using the new console \(p. 66\)](#) and [Task definition parameters \(p. 97\)](#).

For more task definition examples, see [AWS Sample Task Definitions](#) on GitHub.

### Topics

- [Example: Webserver \(p. 189\)](#)
- [Example: splunk log driver \(p. 191\)](#)
- [Example: fluentd log driver \(p. 191\)](#)
- [Example: gelf log driver \(p. 192\)](#)
- [Example: Container dependency \(p. 192\)](#)
- [Windows sample task definitions \(p. 193\)](#)

## Example: Webserver

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

```
{
  "containerDefinitions": [
    {
      "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
<style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' > /usr/local/apache2/htdocs/index.html && httpd-foreground\""
      ],
      "entryPoint": [
        "sh",
        "-c"
      ],
      "essential": true,
      "image": "httpd:2.4",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group" : "/ecs/fargate-task-definition",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "name": "sample-fargate-app",
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
```

```

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

```

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

```

{
  "containerDefinitions": [
    {
      "command": [
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value '<html>
<head> <title>Amazon ECS Sample App</title> <style>body {margin-top: 40px; background-
color: #333;} </style> </head><body> <div style=color:white;text-align:center> <h1>Amazon
ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a
container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"
      ],
      "entryPoint": [
        "powershell",
        "-Command"
      ],
      "essential": true,
      "cpu": 2048,
      "memory": 4096,
      "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/fargate-windows-task-definition",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "name": "sample_windows_app",
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "memory": "4096",
  "cpu": "2048",
  "networkMode": "awsvpc",
  "family": "windows-simple-iis-2019-core",
  "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
  "runtimePlatform": {
    "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"
  }
}

```

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

## 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 \(p. 176\)](#).

```
"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 \(p. 176\)](#).

```
"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. 110).

```
"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: 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. 116).

```
{
  "family": "appmesh-gateway",
  "runtimePlatform": {
    "operatingSystemFamily": "LINUX"
  },
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "envoy",
    "properties": [
      {
        "name": "IgnoredUID",
        "value": "1337"
      },
      {
        "name": "ProxyIngressPort",
        "value": "15000"
      },
      {
        "name": "ProxyEgressPort",
        "value": "15001"
      },
      {
        "name": "AppPorts",
        "value": "9080"
      }
    ]
  }
}
```

```

        "name": "EgressIgnoredIPs",
        "value": "169.254.170.2,169.254.169.254"
    }
]
},
"containerDefinitions": [
    {
        "name": "app",
        "image": "application_image",
        "portMappings": [
            {
                "containerPort": 9080,
                "hostPort": 9080,
                "protocol": "tcp"
            }
        ],
        "essential": true,
        "dependsOn": [
            {
                "containerName": "envoy",
                "condition": "HEALTHY"
            }
        ]
    },
    {
        "name": "envoy",
        "image": "840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-
prod",
        "essential": true,
        "environment": [
            {
                "name": "APPMESH_VIRTUAL_NODE_NAME",
                "value": "mesh/meshName/virtualNode/virtualNodeName"
            },
            {
                "name": "ENVOY_LOG_LEVEL",
                "value": "info"
            }
        ],
        "healthCheck": {
            "command": [
                "CMD-SHELL",
                "echo hello"
            ],
            "interval": 5,
            "timeout": 2,
            "retries": 3
        }
    }
],
"executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
"networkMode": "awsvpc"
}

```

## Windows sample task definitions

The following is a sample task definition to help you get started with Windows containers on Amazon ECS.

### Example Amazon ECS Console Sample Application for Windows

The following task definition is the Amazon ECS console sample application that is produced in the first-run wizard for Amazon ECS; it has been ported to use the `microsoft/iis` Windows container image.

```
{
  "family": "windows-simple-iis",
  "containerDefinitions": [
    {
      "name": "windows_sample_app",
      "image": "mcr.microsoft.com/windows/servercore/iis",
      "cpu": 1024,
      "entryPoint": ["powershell", "-Command"],
      "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value '<html>
<head> <title>Amazon ECS Sample App</title> <style>body {margin-top: 40px; background-
color: #333;} </style> </head><body> <div style=color:white;text-align:center> <h1>Amazon
ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a
container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"],
      "portMappings": [
        {
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "memory": 1024,
      "essential": true
    }
  ],
  "networkMode": "awsvpc",
  "memory": "1024",
  "cpu": "1024"
}
```

## Updating a task definition using the classic console

A *task definition revision* is a copy of the current task definition with the new parameter values replacing the existing ones. All parameters that you do not modify are in the new revision.

To update a task definition, create a task definition revision. If the task definition is used in a service, you must update that service to use the updated task definition.

### 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 \(p. 237\)](#).

## Deregistering a task definition revision

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

You can't use an `INACTIVE` task definition revision to run new tasks or create new services. You also can't update an existing service to reference an `INACTIVE` task definition revision. This is despite that there might be up to a 10-minute window following deregistration where these restrictions have not already 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. Therefore, don't rely on `INACTIVE` task definition revisions persisting beyond the lifecycle of any associated tasks and services.

**To deregister a new task definition (Classic Amazon ECS 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, 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 that you want to deregister.
6. Choose **Actions, Deregister**.
7. Verify the information in the **Deregister task definition** window, and then choose **Deregister** to finish.

# Account settings

Amazon ECS provides account settings, which provide a way to opt in or out of specific features. 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 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 can 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.

**Note**

Federated users assume the account setting of the root user and can't have explicit account settings set for them.

The following account settings are available. The opt in and opt out option must be selected for each account setting separately.

**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. The opt-in status for each resource type determines the ARN format the resource uses. You must opt-in to the new ARN format to use features such as resource tagging for that resource type. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 197\)](#).

Only resources launched after opting in receive the new ARN and resource ID format. 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, 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.

**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 \(p. 308\)](#).

When you opt in to the `containerInsights` account setting, all new clusters have Container Insights enabled by default. You can disable this setting for specific clusters when you create them. You can also change this setting by using the `UpdateClusterSettings` API.

### Dual-stack VPC IPv6

Resource name: `dualStackIPv6`

Amazon ECS supports providing tasks with an IPv6 address in addition to the primary private IPv4 address.

For tasks to receive an IPv6 address, the task must use the `awsvpc` network mode, must be launched in a VPC configured for dual-stack mode, and the `dualStackIPv6` account setting must be enabled. For more information on other requirements, see [Using a VPC in dual-stack mode \(p. 138\)](#).

#### Important

The `dualStackIPv6` account setting can only be changed using either the Amazon ECS API or the AWS CLI. For more information, see [Modifying account settings \(p. 199\)](#).

If you had a running task using the `awsvpc` network mode in an IPv6 enabled subnet between the dates of October 1, 2020 and November 2, 2020, the default `dualStackIPv6` account setting in the Region the task was running in is disabled. If that condition is not met, the default `dualStackIPv6` setting in the Region is enabled.

### Topics

- [Amazon Resource Names \(ARNs\) and IDs \(p. 197\)](#)
- [ARN and resource ID format timeline \(p. 198\)](#)
- [Viewing account settings \(p. 198\)](#)
- [Modifying account settings \(p. 199\)](#)

## 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 ID format is provided on a per-Region basis. Currently, any new account created is opted in by default.

You can opt in or opt out of the new Amazon Resource Name (ARN) and resource ID format at any time. After you 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 user-friendly name, such as a service named `production`. In other cases, you must specify a resource using the Amazon Resource Name (ARN) format. The new ARN format for Amazon ECS tasks, services, and container instances includes the cluster name. For information about opting in to the new ARN format, see [Modifying account settings \(p. 199\)](#).

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

Resource type	ARN
Container instance	<p>Old: <code>arn:aws:ecs:region:aws_account_id:container-instance/container-instance-id</code></p> <p>New: <code>arn:aws:ecs:region:aws_account_id:container-instance/cluster-name/container-instance-id</code></p>
Amazon ECS service	<p>Old: <code>arn:aws:ecs:region:aws_account_id:service/service-name</code></p> <p>New: <code>arn:aws:ecs:region:aws_account_id:service/cluster-name/service-name</code></p>
Amazon ECS task	<p>Old: <code>arn:aws:ecs:region:aws_account_id:task/task-id</code></p> <p>New: <code>arn:aws:ecs:region:aws_account_id:task/cluster-name/task-id</code></p>

### 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 information about opting in to the new resource ID format, see [Modifying account settings \(p. 199\)](#).

## ARN and resource ID format timeline

There is a timeline for the opt-in and opt-out periods for the new Amazon Resource Name (ARN) and resource ID format for Amazon ECS resources. The ARN and resource ID is set at the time of creation and does not change after that. Therefore, opting in or out of the new format does not affect the ARN or resource ID of your existing resources.

The following are the important dates related to this change.

- From now until September 30, 2020 – 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.
- October 1, 2020 - March 31, 2021 – All new accounts are opted in to the new format by default. Any existing accounts that have not explicitly opted out of the new format are also opted in. The ability to opt in and opt out continues to be available on a per-Region basis.
- April 1, 2021 – All accounts will be opted in by default. All new resources created will receive the new format. The ability to opt out will no longer be available.

You can modify your opt-in setting for the new Amazon Resource Name (ARN) and resource ID format at any time between now and April 1, 2021. After you have opted in, any new resources that you create use the new format.

## Viewing account settings

You can use the AWS Management Console and AWS CLI tools to view your account settings.

### Important

The `dualStackIPv6` account setting can only be viewed or changed using the AWS CLI.

#### To view your account settings (Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation bar at the top, 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** 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 (AWS CLI)

- `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 (AWS CLI)

- `list-account-settings` (AWS CLI)

```
aws ecs list-account-settings --principal-arn  
arn:aws:iam:::user/principalName --effective-settings --region us-east-1
```

- `Get-ECSAccountSetting` (AWS Tools for Windows PowerShell)

```
Get-ECSAccountSetting -PrincipalArn arn:aws:iam:::user/principalName -  
EffectiveSetting true -Region us-east-1
```

## Modifying account settings

You can use the AWS Management Console and AWS CLI tools to modify your account settings.

#### To modify account settings (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** 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 (AWS CLI)

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 other account settings. To do this, replace the name parameter with the corresponding account setting.

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

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

### To modify the account settings for your IAM user account (AWS CLI)

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 other account settings. To do this, replace the name parameter with the corresponding account setting.

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

```
Write-ECSAccountSetting -Name serviceLongArnFormat -Value enabled -Force
```

### To modify the account settings for a specific IAM user or IAM role (AWS CLI)

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 other account settings. To do this, replace the name parameter with the corresponding account setting.

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

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

# 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 use the same cluster state information as the Amazon ECS API to make appropriate placement decisions.

Each task that uses the Fargate launch type has its own isolation boundary and doesn't share underlying resources with any other tasks. These resources include the underlying kernel, CPU resources, memory resources, and elastic network interface.

Amazon ECS provides a service scheduler for long-running tasks and applications. It also provides the ability to run tasks manually for batch jobs or single run tasks. Amazon ECS provides one whenever it places tasks on your cluster. You can specify the task placement strategies and constraints for running tasks that best meet your needs. For example, you can specify whether tasks run across multiple Availability Zones or within a single Availability Zone. And, optionally, you can integrate tasks with your own custom or third-party schedulers.

## Service scheduler

The service scheduler is suitable for long running stateless services and applications. The service scheduler ensures that the scheduling strategy that you specify is followed and reschedules tasks when a task fails. For example, if the underlying infrastructure fails, the service scheduler can reschedule tasks.

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 \(p. 213\)](#).
- **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](#) in the *Amazon Elastic Container Service Developer Guide*.

### 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. This might include deploying a new task definition or changing the number of desired tasks that are running. By default, the service scheduler spreads tasks across multiple Availability Zones. However, you can use task placement strategies and constraints to customize task placement decisions. For more information, see [Amazon ECS services \(p. 212\)](#).

## Manually running tasks

The `RunTask` action is suitable for processes such as batch jobs that perform work and then stop. For example, you can have a process call `RunTask` when work comes into a queue. The task pulls work from the queue, performs the work, and then exits. Using `RunTask`, you can allow the default task placement strategy to distribute tasks randomly across your cluster. This minimizes the chances that a single instance gets a disproportionate number of tasks. Alternatively, you can use `RunTask` to customize how the scheduler places tasks using task placement strategies and constraints. For more information, see [Run a standalone task \(p. 202\)](#) 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, you can use the Amazon ECS console to create an EventBridge event. You can run tasks for a backup operation or a log scan. The EventBridge event that you create can run one or more tasks in your cluster at specified times. Your scheduled event can be set to a specific interval (run every *N* minutes, hours, or days). Otherwise, for more complicated scheduling, you can use a `cron` expression. For more information, see [Scheduled tasks \(p. 204\)](#).

#### Contents

- [Run a standalone task \(p. 202\)](#)
- [Scheduled tasks \(p. 204\)](#)
- [AWS Fargate task maintenance \(p. 208\)](#)

## Run a standalone task

We recommend that you deploy your application as a standalone task in some situations. For example, suppose that you're developing an application but you're not ready to deploy it with the service scheduler. If your application is a one-time or periodic batch job, it doesn't make sense to keep running or restart when it finishes.

To deploy your application to run continually or to place it behind a load balancer, create an Amazon ECS service. For more information, see [Amazon ECS services \(p. 212\)](#).

#### Classic console

To run a standalone task using the classic console

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, select the box to the left of the task definition to run.
  - To run an earlier revision of a task definition, select the task definition to view all active revisions. Last, select the revision to run.
3. Choose **Actions, Run Task**.
4. On the **Run Task** page, complete the following steps.
  - a. Choose either a capacity provider strategy or a launch type.
    - To use a **Capacity provider strategy** and choose **Switch to capacity provider strategy**. Then, choose whether your task uses the default capacity provider strategy that's defined for the cluster or a custom capacity provider strategy. A capacity provider must be associated with the cluster to be used in a custom capacity provider strategy. For more information, see [Amazon ECS capacity providers \(p. 78\)](#).
    - To use a **Launch type**, choose **Switch to launch type** and select either **EC2** or **EXTERNAL**. For more information about launch types, see [Amazon ECS launch types \(p. 124\)](#).
  - b. For **Operating system family**, choose the container operating system that the task runs on.
  - c. For **Cluster**, choose the cluster to use.
  - d. For **Number of tasks**, enter the number of tasks to launch with this task definition.
  - e. For **Task group**, enter the name of the task group.
5. If your task definition uses the `awsvpc` network mode, complete these substeps. Otherwise, proceed 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. Tasks don't receive public IP addresses. Therefore, a NAT gateway is required for outbound internet access, and inbound internet traffic is routed through a load balancer.

- c. For **Security groups**, a security group was 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, choose **Edit** and then modify your security group settings. Do the same if you want to choose an existing security group.
6. (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 intend to use the parameter values from your task definition, you don't need to specify overrides. These fields are only used to override the values that are specified in the task definition.

- a. For **Task Role Override**, choose an IAM role for this task to override the task IAM role that's specified in the task definition. For more information, see [IAM roles for tasks \(p. 360\)](#).

Only roles with the `ecs-tasks.amazonaws.com` trust relationship are displayed. For instructions on how to create an IAM role for your tasks, see [Creating an IAM role and policy for your tasks \(p. 361\)](#).

- 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 \(p. 354\)](#).
- c. For **Container Overrides**, choose a container to which to send a command or environment variable override.
  - **For a command override:** For **Command override**, enter the command override to send. If your container definition doesn't specify an `ENTRYPOINT`, the format is a comma-separated list of non-quoted strings.

```
/bin/sh, -c, echo, $DATE
```

If your container definition specifies an `ENTRYPOINT` (such as `sh, -c`), enter a string value for your environment value (without the surrounding double quotation marks (" ")). AWS surrounds the strings with double quotation marks (" ") and passes the string as an argument to the `ENTRYPOINT` command.

```
while true; do echo $DATE > /var/www/html/index.html; sleep 1; done
```

- **For environment variable overrides:** Choose **Add Environment Variable**. For **Key**, enter the name of your environment variable. For **Value**, enter a string value for your environment value (without the surrounding double quotation marks (" ")).

AWS surrounds the strings with double quotation marks (" ") and passes the string to the container in the following format:

```
MY_ENV_VAR="This variable contains a string."
```

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 the `PENDING` to the `STOPPED` status, your task might be stopping because of an error. This is also the case if it displays a `PENDING` status and then disappears from the listed tasks. For more information, see [Checking stopped tasks for errors \(p. 442\)](#) in the troubleshooting section.

Command line

Use the `run-task` command. For more information, see [run-task](#) in the *AWS Command Line Interface Reference*.

## Scheduled tasks

Amazon ECS supports creating scheduled tasks. Scheduled tasks use Amazon EventBridge rules to run tasks either on a schedule or in a response to an EventBridge event.

If you want to run tasks at set intervals, such as a backup operation or a log scan, you can create a scheduled task that runs one or more tasks at specified times. You can specify a regular interval (run every `N` minutes, hours, or days), or for more complicated scheduling, you can use a `cron` expression. For more information, see [Cron expressions and rate expressions](#) in the *Amazon EventBridge User Guide*.

If you want to run tasks that are triggered by an event, there are AWS managed events for services (for example Amazon ECS task and container instance state change events) or you can create a custom event pattern. For more information, see [Event patterns](#) in the *Amazon EventBridge User Guide*.

**Contents**

- [Create a scheduled task \(p. 204\)](#)
- [View your scheduled tasks \(p. 208\)](#)
- [Edit a scheduled task \(p. 208\)](#)

## Create a scheduled task

Scheduled tasks are triggered by Amazon EventBridge rules, which you can create using the EventBridge console. Although you can create a scheduled task in the Amazon ECS console, currently the EventBridge console provides more functionality so the following steps walk you through creating an EventBridge rule that triggers a scheduled task.

**Create a scheduled task (EventBridge console)**

1. Open the Amazon EventBridge console at <https://console.aws.amazon.com/events/>.

2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule.

**Note**

A rule can't have the same name as another rule in the same Region and on the same event bus.

5. For **Event bus**, choose the event bus that you want to associate with this rule. If you want this rule to match events that come from your account, select **AWS default event bus**. When an AWS service in your account emits an event, it always goes to your account's default event bus.
6. Choose how to schedule the task.

To create a rule based on...	Do this...	
Event	<ol style="list-style-type: none"> <li>a. For <b>Rule type</b>, choose <b>Rule with an event pattern</b>.</li> <li>b. Choose <b>Next</b>.</li> <li>c. For <b>Event source</b>, choose <b>AWS events</b>.</li> <li>d. For <b>Event pattern</b>, do one of the following: <ul style="list-style-type: none"> <li>• To use a template to create your event pattern, choose <b>Event pattern form</b> and choose the <b>Event source</b>, <b>AWS service</b>, and <b>Event type</b>. If you choose <b>All Events</b> as the event type, all events emitted by the AWS service will match the rule.</li> <li>To customize the template, choose <b>Custom pattern (JSON editor)</b> and make your changes.</li> <li>• To use a custom event pattern, choose <b>Custom pattern (JSON editor)</b> and create your event pattern.</li> </ul> </li> </ol>	
Schedule	<ol style="list-style-type: none"> <li>a. For <b>Rule type</b>, choose <b>Schedule</b>.</li> <li>b. Choose <b>Next</b>.</li> <li>c. For <b>Schedule pattern</b>, do one of the following: <ul style="list-style-type: none"> <li>• To use a cron expression to define the schedule, choose <b>A fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the</b></li> </ul> </li> </ol>	

To create a rule based on...	Do this...	
	<p><b>first Monday of every month</b> and enter the cron expression.</p> <ul style="list-style-type: none"> <li>To use a rate expression to define the schedule, choose <b>A schedule that runs at a regular rate, such as every 10 minutes</b> and enter the rate expression.</li> </ul>	

7. Choose **Next**.
8. For **Target types**, choose **AWS service**.
9. For **Select a target**, select **ECS task**.
10. For **Cluster**, select an Amazon ECS cluster.
11. For **Task definition**, select a task definition family.
12. For **Task definition revision**, select either **Latest** or **Revision** and select a specific task definition revision to use.
13. For **Count**, specify the desired number of tasks to run.
14. Choose how your scheduled task is distributed across your cluster infrastructure.

Distribution method	Steps	
Capacity provider strategy	<p>a. In the <b>Compute options</b> section, select <b>Capacity provider strategy</b>.</p> <p>b. Choose a strategy:</p> <ul style="list-style-type: none"> <li>To use the cluster's default capacity provider strategy, choose <b>Use cluster default</b>.</li> <li>If your cluster doesn't have a default capacity provider strategy, or to use a custom strategy, choose <b>Use custom, Add capacity provider strategy</b> and define your custom capacity provider strategy by specifying a <b>Base, Capacity provider, and Weight</b>.</li> </ul> <p><b>Note</b> To use a capacity provider in a strategy, the capacity provider must be associated with the cluster. For more information about</p>	

Distribution method	Steps	
	capacity provider strategies, see <a href="#">Amazon ECS capacity providers (p. 78)</a> .	
Launch type	<ol style="list-style-type: none"> <li>In the <b>Compute options</b> section, select <b>Launch type</b>.</li> <li>For <b>Launch type</b>, choose a launch type.</li> <li>(Optional) When the Fargate launch type is specified, for <b>Platform version</b>, specify the platform version to use. If a platform version isn't specified, the <b>LATEST</b> platform version is used by default.</li> </ol>	

15. If the task is hosted on Fargate or uses the `awsvpc` network mode, expand **Configure network configuration** and specify a network configuration.
    - For **Subnets**, specify one or more subnet IDs.
    - For **Security groups**, specify one or more security group IDs.
    - For **Auto-assign public IP**, specify whether to assign a public IP address from your subnet to the task.
  16. (Optional) To specify additional parameters for your tasks, expand **Configure additional properties**.
    - For **Task group**, specify a task group name. The task group name is used to identify a set of related tasks and is used in conjunction with the `spread` task placement strategy to ensure tasks in the same task group are spread out evenly among the container instances in the cluster.
    - For **Tags**, choose **Add tag** to associate key value pair tags for the task.
    - To add tags to use when reviewing cost allocation in your Cost and Usage Report, for **Configure managed tags**, choose **Enable managed tags**. For more information, see [Tagging your resources for billing \(p. 278\)](#).
    - To enable the ECS Exec functionality for the task, for **Configure execute command**, choose **Enable execute command**. For more information, see [Using Amazon ECS Exec for debugging \(p. 433\)](#).
    - To add the tags associated with the task definition to your task, for **Configure propagate tags**, choose **Propagate tags from task definition**. For more information, see [Tagging your resources \(p. 277\)](#).
- Note**  
If you specify a tag with the same key in the **Tags** section, that tag overrides the tag that is propagated from the task definition.
17. For many target types, EventBridge needs permissions to send events to the target. In these cases, EventBridge can create the IAM role needed for your rule to run. Do one of the following:
    - To create an IAM role automatically, choose **Create a new role for this specific resource**.
    - To use an IAM role that you created earlier, choose **Use existing role** and select the existing role from the dropdown.
  18. (Optional) For **Additional settings**, do the following:
    - For **Maximum age of event**, enter a value between one minute (00:01) and 24 hours (24:00).
    - For **Retry attempts**, enter a number between 0 and 185.



- c. For **Dead-letter queue**, choose whether to use a standard Amazon SQS queue as a dead-letter queue. EventBridge sends events that match this rule to the dead-letter queue if they are not successfully delivered to the target. Do one of the following:
  - Choose **None** to not use a dead-letter queue.
  - Choose **Select an Amazon SQS queue in the current AWS account to use as the dead-letter queue** and then select the queue to use from the dropdown.
  - Choose **Select an Amazon SQS queue in an other AWS account as a dead-letter queue** and then enter the ARN of the queue to use. You must attach a resource-based policy to the queue that grants EventBridge permission to send messages to it. For more information, see [Granting permissions to the dead-letter queue](#) in the *Amazon EventBridge User Guide*.
19. Choose **Next**.
20. (Optional) Enter one or more tags for the rule. For more information, see [Amazon EventBridge tags](#) in the *Amazon EventBridge User Guide*.
21. Choose **Next**.
22. Review the details of the rule and choose **Create rule**.

## View your scheduled tasks

Your scheduled tasks can be viewed in the classic Amazon ECS console. You can also view the Amazon EventBridge rules that trigger the scheduled tasks in the EventBridge console.

### To view your scheduled tasks (Amazon ECS console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. Choose the cluster your scheduled tasks are run in.
3. On the **Cluster: *cluster-name*** page, choose the **Scheduled Tasks** tab.
4. All of your scheduled tasks are listed.

## Edit a scheduled task

Your scheduled tasks can be edited in the classic Amazon ECS console. You can also edit the Amazon EventBridge rules that trigger the scheduled tasks in the EventBridge console.

### To edit a scheduled task (Amazon ECS console)

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**.

## AWS Fargate task maintenance

When AWS determines that a security or infrastructure update is needed for an Amazon ECS task hosted on AWS Fargate, the tasks need to be stopped and new tasks launched to replace them.

For tasks that are part of an Amazon ECS service, if there's an issue with the underlying host, AWS stops the task. Moreover, the service scheduler also launches a new task in an attempt to maintain the desired

count for the service. When this occurs, no task retirement notice is sent. However, if there's a security issue with the underlying host or platform version that the task is using, a task retirement notice is sent to your [AWS Health Dashboard](#). The notice is also sent to the email address that's associated with the account. The task retirement notice provides details about the issue, the task retirement date, and what the next steps are. For more information, see [Understanding the task retirement notice \(p. 210\)](#).

For standalone tasks, when there's an issue with the underlying host or a security issue with the platform version that the task is using, AWS sends a task retirement notice to your [AWS Health Dashboard](#). The notice also is sent to the email address associated with the account. The task retirement notice provides details about the issue, the task retirement date, and what the next steps are. For more information, see [Understanding the task retirement notice \(p. 210\)](#).

When a task is stopped in any of the scenarios mentioned here, you can describe the stopped task to retrieve the `stoppedReason` value. The `stoppedReason` containing a `ECS is performing maintenance on the underlying infrastructure hosting the task` message indicates that the task was stopped due to a task maintenance issue.

### Important

To prepare for the task retirement process, we recommend that you test your application behavior by simulating this scenario. You can do this by stopping an individual task in your service to test for resiliency.

The following table describes these scenarios.

Task type	Issue	Action
Standalone task	Host issue	A task retirement notice is sent using your <a href="#">AWS Health Dashboard</a> and email. If no action is taken by the task retirement date, AWS stops the task.
	Security vulnerability	A task retirement notice is sent using your <a href="#">AWS Health Dashboard</a> and email. If no action is taken by the task retirement date, AWS stops the task.
Service task	Host issue	The task is stopped by AWS and the service scheduler launches a new task in an attempt to maintain the desire count for the service. No notification is sent.
	Security vulnerability	A task retirement notice is sent using your <a href="#">AWS Health Dashboard</a> and email. If no action is taken by the task retirement date, AWS stops the task and the service scheduler launches a new task in an attempt to maintain the service's desired count.

## Understanding the task retirement notice

When a task retirement notice is sent, you're notified by email of the pending retirement. An email is sent before the event with the task ID and retirement date. This email is sent to the address that's associated with your account. This is the same email address that you use to log in to the AWS Management Console. If you use an email account that you don't check regularly, you can use the [AWS Health Dashboard](#) to determine if any of your tasks are scheduled for retirement. You can update the contact information for your account on the [Account Settings](#) page.

When a task reaches its scheduled retirement date, it's stopped or terminated by AWS. This is if it hasn't already been stopped. For service tasks, the service scheduler launches a new task to replace the retired task, and then stops the task that will be retired. The service scheduler maintains the service's desired count. For standalone tasks, they're stopped and you're responsible for launching a replacement.

## Working with tasks scheduled for retirement

### Note

This procedure only applies to service tasks. For standalone tasks, simply stop and run new standalone tasks.

To update your service tasks before the retirement date, you can use the following steps. For more information, see [Updating a service \(p. 237\)](#).

### 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 where 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

When you force a new deployment, the scheduler launches new tasks using the patched platform version. Your tasks don't require you select a different platform version in order to update. For more information, see [AWS Fargate platform versions \(p. 58\)](#).

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 Amazon Resource Name (ARN) for the service.

```
aws ecs list-services --cluster cluster_name --region region
```

The output is as follows.

```
{
  "serviceArns": [
    "arn:aws:ecs:region:aws_account_id:service/MyService"
  ]
}
```

2. Update your service that deploys new tasks.

You might need to set the `force-new-deployment` option depending on your deployment type and changed service options. For information about when to set the `force-new-deployment` option, see [update-services](#) in the *AWS CLI Command Reference*.

```
aws ecs update-service --service serviceArn --cluster cluster_name --region region
```

If you're using standalone tasks, then you can start a new task to replace it. For more information, see [Run a standalone task](#) (p. 202).

# Amazon ECS services

You can use an Amazon ECS service to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. If one of your tasks fails or stops, the Amazon ECS service scheduler launches another instance of your task definition to replace it. This helps maintain your desired number of tasks in the service.

You can also optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

## Topics

- [Service scheduler concepts \(p. 212\)](#)
- [Additional service concepts \(p. 213\)](#)
- [Service definition parameters \(p. 213\)](#)
- [Creating an Amazon ECS service \(p. 226\)](#)
- [Updating a service \(p. 237\)](#)
- [Deleting a service \(p. 239\)](#)
- [Amazon ECS Deployment types \(p. 240\)](#)
- [Service load balancing \(p. 252\)](#)
- [Service auto scaling \(p. 263\)](#)
- [Service Discovery \(p. 271\)](#)
- [Service throttle logic \(p. 274\)](#)

## Service scheduler concepts

We recommend that you use the service scheduler for long running stateless services and applications. The service scheduler ensures that the scheduling strategy that you specify is followed and reschedules tasks when a task fails. For example, if the underlying infrastructure fails, the service scheduler reschedules a task. You can use task placement strategies and constraints to customize how the scheduler places and terminates tasks. If a task in a service stops, the scheduler launches a new task to replace it. This process continues until your service reaches your desired number of tasks based on the scheduling strategy that the service uses. The scheduling strategy of the service is also referred to as the *service type*.

The service scheduler includes logic that throttles how often tasks are restarted if tasks repeatedly fail to start. If a task is stopped without having entered a `RUNNING` state, the service scheduler starts to slow down the launch attempts and sends out a service event message. This behavior prevents unnecessary resources from being used for failed tasks before you can resolve the issue. After the service is updated, the service scheduler resumes normal scheduling behavior. For more information, see [Service throttle logic \(p. 274\)](#) and [Service event messages \(p. 450\)](#).

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 \(p. 213\)](#).

- **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](#) in the *Amazon Elastic Container Service Developer Guide*.

**Note**

Fargate tasks do not support the **DAEMON** scheduling strategy.

## Replica

The *replica* scheduling strategy places and maintains the desired number of tasks in your cluster.

For a service that runs tasks on Fargate, when the service scheduler launches new tasks or stops running tasks, the service scheduler attempts to maintain a balance across Availability Zones. You don't need to specify task placement strategies or restraints.

## Additional service concepts

- You can optionally run your service behind a load balancer. For more information, see [Service load balancing](#) (p. 252).
- You can optionally specify a deployment configuration for your service. A deployment is triggered by updating 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. 213).
- You can optionally configure your service to use Amazon ECS service discovery. Service discovery uses the AWS Cloud Map autonaming APIs to manage DNS entries for your service's tasks. This makes them discoverable from within your VPC. For more information, see [Service Discovery](#) (p. 271).
- When you delete a service, if there are still running tasks that require cleanup, the service moves from an **ACTIVE** to a **DRAINING** status, and the service is no longer visible in the console or in the `ListServices` API operation. After all tasks have transitioned to either a **STOPPING** or **STOPPED** status, the service moves from a **DRAINING** to **INACTIVE** status. You can view services in the **DRAINING** or **INACTIVE** status by using the `DescribeServices` API operation. However, in the future, **INACTIVE** services might 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` | `EXTERNAL`

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 \(p. 124\)](#).

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 you want to specify a capacity provider that uses an Auto Scaling group, the capacity provider must already be created. New capacity providers can be created with the `CreateCapacityProvider` API operation.

To use an AWS Fargate capacity provider, specify either the `FARGATE` or `FARGATE_SPOT` capacity providers. The AWS Fargate capacity providers are available to all accounts and only need to be associated with a cluster to be used.

The `PutClusterCapacityProviders` API operation is used to update the list of available capacity providers for a cluster after the cluster is created.

`capacityProvider`

Type: String

Required: Yes

The short name or full Amazon Resource Name (ARN) of the capacity provider.

`weight`

Type: Integer

Valid range: Integers between 0 and 1,000.

Required: No

The weight value designates the relative percentage of the total number of tasks launched that use the specified capacity provider.

For example, assume that you have a strategy that contains two capacity providers and both have a weight of one. When the base is satisfied, the tasks split evenly across the two capacity providers. Using that same logic, assume that you specify a weight of 1 for *capacityProviderA* and a weight of 4 for *capacityProviderB*. Then, for every one task that is run using *capacityProviderA*, four tasks use *capacityProviderB*.

`base`

Type: Integer

Valid range: Integers between 0 and 100,000.

Required: No

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

## Task definition

`taskDefinition`

Type: String

Required: No

The family and revision (`family:revision`) or full Amazon Resource Name (ARN) of the task definition to run in your service. If a revision isn't specified, the latest `ACTIVE` revision of the specified family is used.

A task definition must be specified when using the rolling update (ECS) deployment controller.

## Platform operating system

`platformFamily`

Type: string

Required: Conditional

Default: Linux

This parameter is required for Amazon ECS services hosted on Fargate.

This parameter is ignored for Amazon ECS services hosted on Amazon EC2.

The operating system on the containers that runs the service. The valid values are `LINUX`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`.

The `platformFamily` value for every task that you specify for the service must match the service `platformFamily` value. For example, if you set the `platformFamily` to `WINDOWS_SERVER_2019_FULL`, the `platformFamily` value for all the tasks must be `WINDOWS_SERVER_2019_FULL`.

## Platform version

`platformVersion`

Type: String

Required: No



The platform version on which your tasks in the service are running. A platform version is only specified for tasks using the Fargate launch type. If one is not specified, the latest version (`LATEST`) is used by default.

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 \(p. 58\)](#).

## 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`

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 \(p. 212\)](#).

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 \(p. 213\)](#).
- **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](#) in the *Amazon Elastic Container Service Developer Guide*.

**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. It is expressed as a percentage of the `desiredCount` that is rounded down to the nearest integer. You can use this parameter to define the deployment batch size. For example, if your service is using the `REPLICA` service scheduler and has a `desiredCount` of four tasks and a `maximumPercent` value of 200%, the scheduler might start four new tasks before stopping the four older tasks. This is provided that the cluster resources required to do this are available. The default `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%. This is the default value.

The maximum number of tasks during a deployment is the `desiredCount` multiplied by the `maximumPercent/100`, rounded down to the nearest integer value.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and tasks that use the EC2 launch type, the **maximum percent** value is set to the default value 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 isn't used, although it's returned when describing your service.

`minimumHealthyPercent`

Type: Integer

Required: No

If a service is using the rolling update (ECS) deployment type, the `minimumHealthyPercent` represents a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment. This is expressed as a percentage of the `desiredCount` that is rounded up to the nearest integer. You can use this parameter to deploy without using additional cluster capacity. For example, if your service has a `desiredCount` of four tasks and a `minimumHealthyPercent` of 50%, the service scheduler might stop two existing tasks to free up cluster capacity before starting two new tasks.

For services that *do not* use a load balancer, consider the following:

- A service is considered healthy if all essential containers within the tasks in the service pass their health checks.
- If a task has no essential containers with a health check defined, the service scheduler waits for 40 seconds after a task reaches a `RUNNING` state before the task is counted towards the minimum healthy percent total.
- If a task has one or more essential containers with a health check defined, the service scheduler waits for the task to reach a healthy status before counting it towards the minimum healthy percent total. A task is considered healthy when all essential containers within the task have passed their health checks. The amount of time the service scheduler can wait for is determined by the container health check settings. For more information, see [Health check \(p. 104\)](#).

For services that *do* use a load balancer, consider the following:

- If a task has no essential containers with a health check defined, the service scheduler waits for the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.
- If a task has an essential container with a health check defined, the service scheduler waits for both the task to reach a healthy status and the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.

The default value for a replica service for `minimumHealthyPercent` is 100%. The default `minimumHealthyPercent` value for a service using the `DAEMON` service schedule is 0% for the AWS CLI, the AWS SDKs, and the APIs and 50% for the AWS Management Console.

The minimum number of healthy tasks during a deployment is the `desiredCount` multiplied by the `minimumHealthyPercent`/100, rounded up to the nearest integer value.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and is running tasks that use the EC2 launch type, the **minimum healthy percent** value is set to the default value 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 a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and is running tasks that use the Fargate launch type, the minimum healthy percent value is not used, although it is returned when describing your service.

## Deployment controller

`deploymentController`

Type: Object

Required: No

The deployment controller to use for the service. If no deployment controller is specified, the ECS controller is used. For more information, see [Amazon ECS Deployment types \(p. 240\)](#).

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

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's specified with the `field` parameter. For example, if you binpack on memory, a task is placed on the instance with the least amount of remaining memory but still enough to run the task.

field

Type: String

Required: No

The field to apply the placement strategy against. For the `spread` placement strategy, valid values are `instanceId` (or `host`, which has the same effect), or any platform or custom attribute that's applied to a container instance, such as `attribute:ecs.availability-zone`. For the `binpack` placement strategy, valid values are `cpu` and `memory`. For the `random` placement strategy, this field is not used.

## Tags

`tags`

Type: Array of objects

Required: No

The metadata that you apply to the service to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see [Tagging your Amazon ECS resources \(p. 276\)](#).

`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).

`enableECSTags`

Type: Boolean

Valid values: `true` | `false`

Required: No

Specifies whether to use Amazon ECS managed tags for the tasks in the service. If no value is specified, the default value is `false`. For more information, see [Tagging your resources for billing \(p. 278\)](#).

`propagateTags`

Type: String

Valid values: `TASK_DEFINITION` | `SERVICE`

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation or task creation, use the `TagResource` API action.

## Network configuration

`networkConfiguration`

Type: Object

Required: No

The network configuration for the service. This parameter is required for task definitions that use the `awsvpc` network mode to receive their own Elastic Network Interface, and it isn't supported for other network modes. If using the Fargate launch type, the `awsvpc` network mode is required. For more information, see [Fargate Task Networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

`awsvpcConfiguration`

Type: Object

Required: No

An object representing the subnets and security groups for a task or service.

`subnets`

Type: Array of strings

Required: Yes

The subnets that are associated with the task or service. There is a limit of 16 subnets that can be specified according to `awsvpcConfiguration`.

`securityGroups`

Type: Array of strings

Required: No

The security groups associated with the task or service. If you don't specify a security group, the default security group for the VPC is used. There's a limit of five security groups that can be specified based on `awsvpcConfiguration`.

`assignPublicIP`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether the task's elastic network interface receives a public IP address. If no value is specified, the default value of `DISABLED` is used.

`healthCheckGracePeriodSeconds`

Type: Integer

Required: No

The period of time, in seconds, that the Amazon ECS service scheduler 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.

If you do not use an Elastic Load Balancing, we recommend that you use the `startPeriod` in the task definition health check parameters. For more information, see [Health check](#).

#### `loadBalancers`

Type: Array of objects

Required: No

A load balancer object representing the load balancers to use with your service. For services that use an Application Load Balancer or Network Load Balancer, there's a limit of five target groups that you can attach to a service.

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.

For Application Load Balancers and Network Load Balancers, this object must contain the load balancer target group ARN, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance and port combination is registered as a target in the target group specified.

#### `targetGroupArn`

Type: String

Required: No

The full Amazon Resource Name (ARN) of the Elastic Load Balancing target group that's associated with a service.

A target group ARN is only specified when using an Application Load Balancer or Network Load Balancer. If you're using a Classic Load Balancer, omit the target group ARN.

#### `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're using an Application Load Balancer or a Network Load Balancer, omit the load balancer name parameter.

`containerName`

Type: String

Required: No

The name of the container (as it appears in a container definition) to associate with the load balancer.

`containerPort`

Type: Integer

Required: No

The port on the container to associate with the load balancer. This port must correspond to a `containerPort` in the task definition used by tasks in the service. For tasks that use the EC2 launch type, the container instance must allow inbound traffic on the `hostPort` of the port mapping.

`role`

Type: String

Required: No

The short name or full ARN of the IAM role that allows Amazon ECS to make calls to your load balancer on your behalf. This parameter is only permitted if you are using a load balancer with a single target group for your service, and your task definition does not use the `awsvpc` network mode. If you specify the `role` parameter, you must also specify a load balancer object with the `loadBalancers` parameter.

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 \(p. 348\)](#).

`serviceRegistries`

Type: Array of objects

Required: No

The details of the service discovery configuration for your service. For more information, see [Service Discovery \(p. 271\)](#).

`registryArn`

Type: String

Required: No

The Amazon Resource Name (ARN) of the service registry. The currently supported service registry is AWS Cloud Map. For more information, see [Working with Services](#) in the *AWS Cloud Map Developer Guide*.



`port`

Type: Integer

Required: No

The port value that's used if your service discovery service specified an SRV record. This field is required if both the `awsvpc` network mode and SRV records are used.

`containerName`

Type: String

Required: No

The container name value to be used for your service discovery service. This value is specified in the task definition. If the task definition that your service task specifies uses the `bridge` or `host` network mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition that your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

`containerPort`

Type: Integer

Required: No

The port value to be used for your service discovery service. This value is specified in the task definition. If the task definition your service task specifies uses the `bridge` or `host` network mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

## Client token

`clientToken`

Type: String

Required: No

The unique, case-sensitive identifier that you provide to ensure the idempotency of the request. It can be up to 32 ASCII characters long.

## 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
},
"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 an Amazon ECS service

When you create an Amazon ECS service, you specify the parameters that define what makes up your service and how your service behaves. These parameters create a service definition. For more information, see [Service definition parameters \(p. 213\)](#).

For services that are hosted on Fargate or Amazon EC2 instances, you can optionally configure an Elastic Load Balancing load balancer to distribute traffic across the containers in your service. For more information, see [Service load balancing \(p. 252\)](#).

### Note

When using a load balancer with services that are hosted on Amazon EC2 instances, verify that your instances can receive traffic from your load balancers. You can allow traffic to all ports on your instances from your load balancer's security group. This ensures that traffic can reach any containers that use dynamically assigned ports.

## Creating a service using the classic Amazon ECS console

### Important

Amazon ECS has provided a new console experience for creating a service. For more information, see [Creating a service using the new console \(p. 72\)](#).

If you're creating a Windows service for the Fargate launch type, you must use the classic console.

The Amazon ECS console provides a create service wizard that guides you through each step to create a service. Use the following pages explain each step in more detail.

### Topics

- [Step 1: Configuring basic service parameters \(p. 226\)](#)
- [Step 2: Configure a network \(p. 228\)](#)
- [Step 3: Configuring your service to use a load balancer \(p. 229\)](#)
- [Step 4: Configuring your service to use Service Discovery \(p. 233\)](#)
- [Step 5: Configuring your service to use Service Auto Scaling \(p. 234\)](#)
- [Step 6: Review and create your service \(p. 236\)](#)

## Step 1: Configuring basic service parameters

### Important

Amazon ECS has provided a new console experience for creating a service. For more information, see [Creating a service using the new console \(p. 72\)](#).

All services require some configuration parameters that define the service, such as the task definition to use, which cluster that the service runs on, how many tasks are placed for the service. This is called the *service definition*. For more information about the parameters defined in a service definition, see [Service definition parameters \(p. 213\)](#).

This procedure covers how to create a service and covers the service definition parameters that are required. After you configured these parameters, you can create your service or move on to the procedures for optional service definition configuration. For example, you can move on to configuring your service to use a load balancer.

### Note

If your cluster is configured with a default capacity provider strategy, you can only create a service using the default capacity provider strategy when using the console. Likewise, if no

default capacity provider is defined, you can only use a launch type when creating a service using the console. It's 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 to create your service from.
4. On the **Task Definition name** page, select the revision of the task definition to create your service from.
5. Review the task definition, and choose **Actions, Create Service**.
6. On the **Configure service** page, complete the following steps.
  - a. Choose either a capacity provider strategy or a launch type.
    - To use a **Capacity provider strategy**, first choose **Switch to capacity provider strategy**. Next, choose whether your service uses the default capacity provider strategy that's defined for the cluster or a custom capacity provider strategy. A capacity provider must already be associated with the cluster to be used in a custom capacity provider strategy. For more information, see [Amazon ECS capacity providers \(p. 78\)](#).
    - To use a **Launch type**, choose **Switch to launch type** and select **FARGATE**, **EC2**, or **EXTERNAL**. For more information about launch types, see [Amazon ECS launch types \(p. 124\)](#).
  - b. For **Platform operating system**, if you chose the Fargate launch type, then select the platform operating system (for example, **LINUX**).
  - c. For **Platform version**, if you chose a Fargate capacity provider or the Fargate launch type, then select the platform version to use.

#### Note

When the **LATEST** platform version is selected, we validate the operating system that was specified for the task, and then set the appropriate platform version. If the Operating System is set to `Windows-Server-2019-Full` or `Windows-Server-2019-Core`, the `1.0.0` platform is used. If the operating system is Linux, the `1.4.0` platform version is used.

- d. **Cluster**: Select the cluster to create your service in.
- e. **Service name**: Type a name for your service. It must be unique.
- f. **Service type**: Select a scheduling strategy for your service. For more information, see [Service scheduler concepts \(p. 212\)](#).
- g. **Number of tasks**: If you chose the `REPLICA` service type, enter 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 doesn't apply if your task definition uses dynamic host port mappings with the `bridge` network mode. For more information, see [portMappings \(p. 102\)](#).

- h. If you're using the **Rolling update** deployment type, fill out the following deployment configuration parameters. For more information about how these parameters are used, see [Deployment configuration \(p. 217\)](#).
  - **Minimum healthy percent**: Specify a lower limit for the number of tasks that your service must remain in the `RUNNING` state during a deployment. Specify the number as a percentage of the desired number of tasks. This number must be a whole number.

- **Maximum percent:** Specify an upper limit for the number of tasks that your service allows in the `RUNNING` or `PENDING` state during a deployment. Specify the number as a percentage of the desired number of tasks. This number must be a whole number.
7. For **Deployment circuit breaker**, choose the deployment circuit breaker logic. For more information, see [the section called "Using the deployment circuit breaker" \(p. 241\)](#).
  8. On the **Deployments** page, complete the following steps.
    - a. For **Deployment type**, choose whether your service uses a rolling update deployment or a blue/green deployment using AWS CodeDeploy. For more information, see [Amazon ECS Deployment types \(p. 240\)](#).
    - b. If you selected the blue/green deployment type, complete the following steps:
      - i. 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 \(p. 243\)](#)
      - ii. For **Service role for CodeDeploy** choose the IAM service role for AWS CodeDeploy. For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 365\)](#)
  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 \(p. 228\)](#).

## Step 2: Configure a network

### Important

Amazon ECS has a new console experience for creating a service. For more information, see [Creating a service using the new console \(p. 72\)](#).

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 uses the `bridge`, `host`, or `none` network modes, you can move on to the next step, [Step 3: Configuring your service to use a load balancer \(p. 229\)](#).

For tasks that are hosted on Amazon EC2 instances, the `awsvpc` network mode doesn't provide task ENIs with public IP addresses. To access the internet, tasks that are hosted on Amazon EC2 instances can be launched in a private subnet that's configured to use a NAT gateway. For more information, see [NAT Gateways](#) in the *Amazon VPC User Guide*. Inbound network access must be from within the VPC using the

private IP address or DNS hostname, or routed through a load balancer from within the VPC. Tasks that are launched within public subnets don't have internet access.

### To configure VPC and security group settings for your service

1. If you didn't do so already, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 226\)](#).
2. For **Cluster VPC**, if you're hosting tasks on Amazon EC2 instances, choose the VPC that your instances are in. If you're hosting tasks on Fargate, select the VPC that the Amazon ECS on Fargate tasks use. Ensure that the VPC that you choose isn't configured to require dedicated hardware tenancy. This feature isn't supported by Fargate.
3. For **Subnets**, choose the available subnets for your service task placement.
4. For **Security groups**, a security group was created for your service's tasks. This security group allows HTTP traffic access 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. For tasks on Fargate, 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're configuring your service to use a load balancer or if you're using the blue/green deployment type, continue to [Step 3: Configuring your service to use a load balancer \(p. 229\)](#). If you aren't 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 \(p. 234\)](#).

## Step 3: Configuring your service to use a load balancer

### Important

Amazon ECS has a new console experience for creating a service. For more information, see [Creating a service using the new console \(p. 72\)](#).

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 aren't 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 \(p. 233\)](#).

If you configured an available Elastic Load Balancing load balancer, 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 \(p. 256\)](#).

### 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 \(p. 229\)](#)
- [Configuring a load balancer for the blue/green deployment type \(p. 231\)](#)

## 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 (about 68 years). 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 didn't do so already, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 226\)](#).
2. For **Health check grace period**: Enter the period of time, in seconds, that the Amazon ECS service scheduler ignores 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 didn't do so already, follow the basic service creation procedures in [Step 1: Configuring basic service parameters \(p. 226\)](#).
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. With host port mapping allowed, you can 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. With host port mapping allowed, you can 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 aren't supported.

We recommend that you use Application Load Balancers for your Amazon ECS services. That way, you can use the advanced features of Application Load Balancer.

3. For **Select IAM role for service**, choose **Create new role** to create the Amazon ECS service-linked role or select your existing service-linked role.
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 that 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 \(p. 230\)](#). If you've chosen a Network Load Balancer, follow the steps in [To configure a Network Load Balancer \(p. 231\)](#).

### 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 distributes 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 \(p. 256\)](#) (if applicable). Alternatively, 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 \(p. 256\)](#) (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 doesn't 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 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 that the load balancer sends health check pings to.
5. When you're 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 \(p. 259\)](#) (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 \(p. 259\)](#) (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 that the load balancer sends health check pings to.
5. When you're finished configuring your Network 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 didn't do so already, follow the procedures to create the service in [Step 1: Configuring basic service parameters \(p. 226\)](#).
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. With host port mapping, you can 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. With host port mapping, you can 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. That way, you can use all of the Application Load Balancer features.

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 that you selected earlier are visible here.
4. The next step depends on the load balancer type for your service. If you chose an Application Load Balancer, follow the steps in [To configure an Application Load Balancer \(p. 230\)](#). If you chose a Network Load Balancer, follow the steps in [To configure a Network Load Balancer \(p. 231\)](#).

### 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 distributes 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 \(p. 256\)](#) (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 task set. 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 task set 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 \(p. 256\)](#) (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. 233).

### 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 a Network Load Balancer](#) (p. 259) (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. 259) (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. 233).

## Step 4: Configuring your service to use Service Discovery

Your Amazon ECS service can optionally use service discovery integration, which allows your service to be discoverable via DNS. For more information, see [Service Discovery](#) (p. 271).

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. 234).

### 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. 226).
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. 234).

## 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. 265) (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. 269)—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. 263).

### 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. 226).
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 turn off 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.

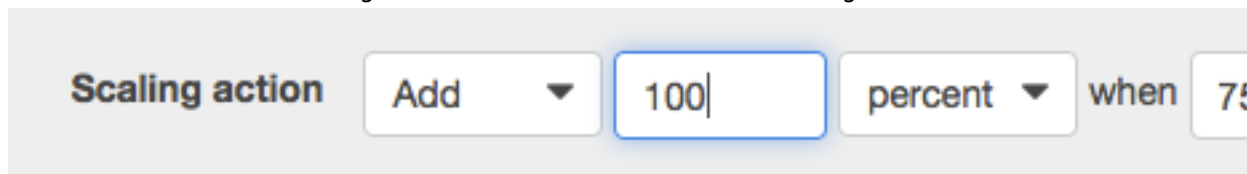
### To configure step scaling policies for your service

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. 263).

- 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%.



Scaling action   Add   100   percent   when   75

6. (Optional) You can repeat [Step 5 \(p. 236\)](#) 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 \(p. 235\)](#) through [Step 8 \(p. 236\)](#) for the **Scale in** policy and choose **Save**.
10. Choose **Next step** to proceed and navigate to [Step 6: Review and create your service \(p. 236\)](#).

## 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 can use the AWS CLI or SDK to modify the load balancer configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*. 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. A service using a Linux platform version can't be updated to use a Windows platform version and vice versa. 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.

#### Topics

- [Updating a service using the classic console \(p. 238\)](#)

## Updating a service using the classic console

### Important

Amazon ECS has provided a new console experience for updating a service. For more information, see [Updating a service using the new console \(p. 73\)](#).

### 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.

### Note

A service using an Auto Scaling group capacity provider can't be updated to use a Fargate capacity provider and vice versa.

A service using a Linux platform version can't be updated to use a Windows platform version and vice versa.

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. 234).
  - 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.

### Classic console

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**.

### 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.

- 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>
```

- Retrieve the ID of the service discovery namespace to delete.

```
aws servicediscovery list-namespaces --region <region_name>
```

- 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.

You can view information about the service deployment type on the service details page, or by using the `describe-services` API. For more information, see [DescribeServices](#) in the *Amazon Elastic Container Service API Reference*.

### Topics

- [Rolling update \(p. 240\)](#)
- [Blue/Green deployment with CodeDeploy \(p. 243\)](#)
- [External deployment \(p. 247\)](#)

## Rolling update

When the *rolling update* (ECS) deployment type is used for your service, when a new service deployment is started the Amazon ECS service scheduler replaces the currently running tasks with new tasks. The number of tasks that Amazon ECS adds or removes from the service during a rolling update is controlled by the deployment configuration. The deployment configuration consists of the `minimumHealthyPercent` and `maximumPercent` values which are defined when the service is created, but can also be updated on an existing service.

The `minimumHealthyPercent` represents the lower limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for the service. This value is rounded up. For example if the minimum healthy percent is 50 and the desired task count is four, then the scheduler can stop two existing tasks before starting two new tasks. Likewise, if the minimum healthy percent is 75% and the desired task count is two, then the scheduler can't stop any tasks due to the resulting value also being two.

The `maximumPercent` represents the upper limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for a service. This value is rounded down. For example if the maximum percent is 200 and the desired task count is four then the scheduler can start four new tasks before stopping four existing tasks. Likewise, if the maximum percent is 125 and the desired task count is three, the scheduler can't start any tasks due to the resulting value also being three.

### Important

When setting a minimum healthy percent or a maximum percent, you should ensure that the scheduler can stop or start at least one task when a deployment is triggered. If your service has a deployment that is stuck due to an invalid deployment configuration, a service event message will be sent. For more information, see [service \(\*service-name\*\) was unable to stop or start tasks during a deployment because of the service deployment configuration](#). Update the `minimumHealthyPercent` or `maximumPercent` value and try again. (p. 452).

When a new service deployment is started or when a deployment is completed, Amazon ECS sends a service deployment state change event to EventBridge. This provides a programmatic way to monitor the status of your service deployments. For more information, see [Service deployment state change events](#) (p. 304).

To create a new Amazon ECS service that uses the rolling update deployment type, see [Creating an Amazon ECS service](#) (p. 226).

## Using the deployment circuit breaker

By default, when a service using the rolling update deployment type starts a new deployment, the service scheduler will launch new tasks until the desired count is reached. You can optionally use deployment circuit breaker logic on the service, which will cause the deployment to transition to a failed state if it can't reach a steady state. The deployment circuit breaker logic can also trigger Amazon ECS to roll back to the last completed deployment upon a deployment failure.

The following `create-service` AWS CLI example shows how to create a Linux service when the deployment circuit breaker enabled with rollback.

```
aws ecs create-service \
  --service-name MyService \
  --deployment-controller type=ECS \
  --desired-count 2 \
  --deployment-configuration "deploymentCircuitBreaker={enable=true,rollback=true}" \
  --task-definition sample-fargate:1 \
  --launch-type FARGATE \
  --platform-os LINUX \
  --platform-version 1.4.0 \
  --network-configuration
  "awsVpcConfiguration={subnets=[subnet-12344321],securityGroups=[sg-12344321],assignPublicIp=ENABLED}"
```

The following should be considered when enabling the deployment circuit breaker logic on a service.

- The deployment circuit breaker is only supported for Amazon ECS services that use the rolling update (ECS) deployment controller and don't use a Classic Load Balancer.
- If a service deployment has at least one successfully running task, the circuit breaker logic will not trigger regardless of the deployment having any previous or future failed tasks.
- There are two new parameters added to the response of a `DescribeServices` API action that provide insight into the state of a deployment, the `rolloutState` and `rolloutStateReason`. When a new deployment is started, the rollout state begins in an `IN_PROGRESS` state. When the service reaches a steady state, the rollout state transitions to `COMPLETED`. If the service fails to reach a steady state and circuit breaker is enabled, the deployment will transition to a `FAILED` state. A deployment in a `FAILED` state won't launch any new tasks.
- In addition to the service deployment state change events Amazon ECS sends for deployments that have started and have completed, Amazon ECS also sends an event when a deployment with circuit breaker enabled fails. These events provide details about why a deployment failed or if a deployment was started because of a rollback. For more information, see [Service deployment state change events](#) (p. 304).

- If a new deployment is started because a previous deployment failed and rollback was enabled, the `reason` field of the service deployment state change event will indicate the deployment was started because of a rollback.

## Failure threshold

The deployment circuit breaker calculates the threshold value, and then uses the value to determine when to move the deployment to a `FAILED` state.

The deployment circuit breaker has a minimum threshold of 10 and a maximum threshold of 200, and uses the values in the following formula to determine the deployment failure.

$$\text{Minimum threshold} \leq 0.5 * \text{desired task count} \Rightarrow \text{maximum threshold}$$

When the result of the calculation is less than the minimum of 10, the failure threshold is set to 10. When the result of the calculation is greater than the maximum of 200, the failure threshold is set to 200.

### Note

You cannot change either of the threshold values.

There are two stages for the deployment status check.

1. The deployment circuit breaker monitors tasks that are part of the deployment and checks for tasks that are in the `RUNNING` state. The scheduler ignores the failure criteria when a task in the current deployment is in the `RUNNING` state and proceeds to the next stage. When tasks fail to reach in the `RUNNING` state, the deployment circuit breaker increases the failure count by one. When the failure count equals the threshold, the deployment is marked as `FAILED`.
2. This stage is entered when there are one or more tasks in the `RUNNING` state. The deployment circuit breaker performs health checks on the following resources for the tasks in the current deployment:
  - Elastic Load Balancing load balancers
  - AWS Cloud Map service
  - Amazon ECS container health checks

When a health check fails for the task, the deployment circuit breaker increases the failure count by one. When the failure count equals the threshold, the deployment is marked as `FAILED`.

The following table provides some examples.

Desired task count	Calculation	Threshold
1	$10 \leq 0.5 * 1 \Rightarrow 200$	10 (the calculated value is less than the minimum)
25	$10 \leq 0.5 * 25 \Rightarrow 200$	13 (the value is rounded up)
400	$10 \leq 0.5 * 400 \Rightarrow 200$	200
800	$10 \leq 0.5 * 800 \Rightarrow 200$	200 (the calculated value is greater than the maximum)

For additional examples about using the rollback option, see [Announcing Amazon ECS deployment circuit breaker](#).

## 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.ECSLinear10PercentEvery1Minute	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.

Deployment configuration	Description
<code>CodeDeployDefault.ECSCanary10percent15minutes</code>	Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed 15 minutes later.
<code>CodeDeployDefault.ECSAllAtOnce</code>	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.
- When a Network Load Balancer is used, only the `CodeDeployDefault.ECSAllAtOnce` deployment configuration is supported.
- For services configured to use service auto scaling and the blue/green deployment type, auto scaling is not blocked during a deployment but the deployment may fail under some circumstances. The following describes this behavior in more detail.
  - If a service is scaling and a deployment starts, the green task set is created and CodeDeploy will wait up to an hour for the green task set to reach steady state and won't shift any traffic until it does.
  - If a service is in the process of a blue/green deployment and a scaling event occurs, traffic will continue to shift for 5 minutes. If the service doesn't reach steady state within 5 minutes, CodeDeploy will stop the deployment and mark it as failed.
  - If a service is in the process of a blue/green deployment and a scaling event occurs, the desired task count might be set to an unexpected value. This is caused by auto scaling considering the running task count as current capacity, which is twice the appropriate number of tasks being used in the desired task count calculation.
- 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 an Amazon ECS service \(p. 226\)](#).

To update an existing Amazon ECS service that is using the blue/green deployment type, see [Updating a service \(p. 237\)](#).

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-< <i>cluster</i> [ : 47 ]>-< <i>service</i> [ : 47 ]>
Deployment group name	DgpECS-< <i>cluster</i> [ : 47 ]>-< <i>service</i> [ : 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 <b>Reroute traffic immediately</b> . You can change it on the CodeDeploy console or by updating the <code>TrafficRoutingConfig</code> . For more information, see <a href="#">CreateDeploymentConfig</a> in the <i>AWS CodeDeploy API Reference</i> .
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 <code>CodeDeployDefault.ECSAllAtOnce</code> by 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.

Resource	Default Setting
Automatic rollback configuration	If a deployment fails, the automatic rollback settings are configured to roll it back.

To view details of an Amazon ECS service using the blue/green deployment type, use the **Deployments** tab on the Amazon ECS console.

To view the details of a CodeDeploy deployment group in the CodeDeploy console, see [View Deployment Group Details with CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

To modify the settings for a CodeDeploy deployment group in the CodeDeploy console, see [Change Deployment Group Settings with CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

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. 341).

```
{
  "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 \(p. 365\)](#).

The [Create Service Example \(p. 338\)](#) and [Update Service Example \(p. 339\)](#) 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 `DeleteTaskSet`). 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, 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 \(p. 276\)](#).

`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).

`enableECSTags`

Specifies whether to use Amazon ECS managed tags for the tasks within the service. For more information, see [Tagging your resources for billing \(p. 278\)](#).

#### `propagateTags`

Type: String

Valid values: `TASK_DEFINITION` | `SERVICE`

Required: No

Specifies whether to copy the tags from the task definition or the service to the tasks in the service. If no value is specified, the tags are not copied. Tags can only be copied to the tasks within the service during service creation. To add tags to a task after service creation or task creation, use the `TagResource` API action.

#### `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.

If you do not use an Elastic Load Balancing, we recommend that you use the `startPeriod` in the task definition health check parameters. For more information, see [Health check](#).

#### `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. 212).

#### `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": ""
        }
    ],
    "enableECSTags": true,
    "propagateTags": "TASK_DEFINITION"
}

```

2. Create an initial task set. The task set contains the following details about your service:

**taskDefinition**

The task definition for the tasks in the task set to use.

**launchType**

Type: String

Valid values: EC2 | FARGATE | EXTERNAL

Required: No

The launch type on which to run your service. If a launch type is not specified, EC2 is used by default. For more information, see [Amazon ECS launch types \(p. 124\)](#).

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 \(p. 58\)](#).

#### 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 [Fargate Task Networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

#### serviceRegistries

The details of the service discovery registries to assign to this service. For more information, see [Service Discovery](#) (p. 271).

#### 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.

### Note

When you use tasks sets, all the tasks in the set must all be configured to use Elastic Load Balancing or to not use Elastic Load Balancing.

Amazon ECS services hosted on AWS Fargate support the Application Load Balancer and Network Load Balancer load balancer types. Application Load Balancers are used to route HTTP/HTTPS (or layer 7) traffic. Network Load Balancers are used to route TCP or UDP (or layer 4) traffic. For more information, see [Load balancer types \(p. 254\)](#).

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.
- 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).

With your load balancer, you pay only for what you use. For more information, see [Elastic Load Balancing pricing](#).

#### Topics

- [Service load balancing considerations \(p. 253\)](#)
- [Load balancer types \(p. 254\)](#)
- [Creating a load balancer \(p. 256\)](#)
- [Registering multiple target groups with a service \(p. 261\)](#)

## 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:

- Amazon ECS requires the service-linked IAM role which provides the permissions needed to register and deregister targets with your load balancer when tasks are created and stopped. For more information, see [Service-linked role for Amazon ECS \(p. 348\)](#).
- For services that use an Application Load Balancer or Network Load Balancer, you cannot attach more than five target groups to a service.
- When you create a target group for your service, you must choose `ip` as the target type, not `instance`.
- If your service uses an Application Load Balancer and 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*.
- 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 can use the AWS CLI or SDK to modify the load balancer configuration. For information about how to modify the configuration, see [UpdateService](#) in the *Amazon Elastic Container Service API Reference*. 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.
- 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 requests and health checks in the target's security group.
- Using a Network Load Balancer to route UDP traffic to your Amazon ECS tasks on Fargate require the task to use platform version 1.4.0 (Linux) or 1.0.0 (Windows).
- Minimize errors in your client applications by setting the `stopTimeout` in the task definition longer than the target group deregistration delay, which should be longer than your client connection timeout. See the Builders Library for more information on recommended client configuration [here](#).

Also, the Network Load Balancer target group attribute for connection termination closes all remaining connections after the deregistration time. This can cause clients to display undesired error messages, if the client does not handle them.

- If you are experiencing problems with your load balancer-enabled services, see [Troubleshooting service load balancers \(p. 454\)](#).

## Load balancer types

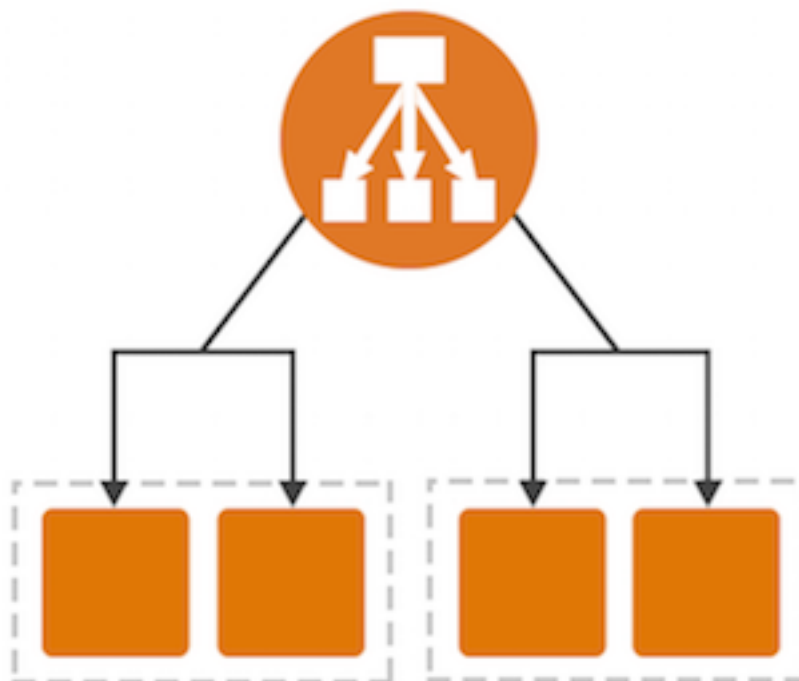
Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. Amazon ECS services can use these types of load balancer. Application Load Balancers are used to route HTTP/HTTPS (or Layer 7) traffic. Network Load Balancers and Classic Load Balancers are used to route TCP (or Layer 4) traffic.

### Topics

- [Application Load Balancer \(p. 254\)](#)
- [Network Load Balancer \(p. 255\)](#)
- [Gateway Load Balancers \(p. 255\)](#)

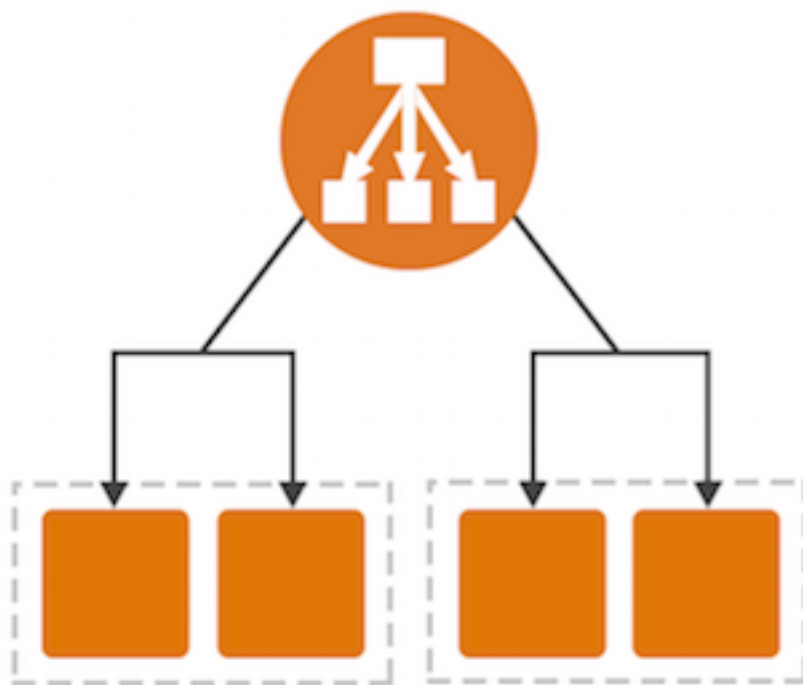
## Application Load Balancer

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](#).



## Gateway Load Balancers

Gateway Load Balancers allow you to deploy, scale, and manage virtual appliances, such as firewalls, intrusion detection and prevention systems, and deep packet inspection systems. It combines a transparent network gateway (that is, a single entry and exit point for all traffic) and distributes traffic while scaling your virtual appliances with the demand. A Gateway Load Balancer operates at the third layer of the Open Systems Interconnection (OSI) model, the network layer. It listens for all IP packets across all ports and forwards traffic to the target group that's specified in the listener rule. It maintains stickiness of flows to a specific target appliance using 5-tuple (for TCP/UDP flows) or 3-tuple (for non-TCP/UDP flows). The Gateway Load Balancer and its registered virtual appliance instances exchange application traffic using the GENEVE protocol on port 6081. It supports a maximum transmission unit (MTU) size of 8500 bytes. Gateway Load Balancers use Gateway Load Balancer endpoints to securely exchange traffic across VPC boundaries. A Gateway Load Balancer endpoint is a VPC endpoint that provides private connectivity between virtual appliances in the service provider VPC and application



servers in the service consumer VPC. You deploy the Gateway Load Balancer in the same VPC as the virtual appliances. You register the virtual appliances with a target group for the Gateway Load Balancer. For more information, see the [Gateway Load Balancers User Guide](#).

## 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-linked role created. For more information, see [Service-linked role for Amazon ECS](#) (p. 348).

### Topics

- [Creating an Application Load Balancer](#) (p. 256)
- [Creating a Network Load Balancer](#) (p. 259)

## Creating an Application Load Balancer

This section walks you through the process of creating an Application Load Balancer in the AWS Management Console. For information about how to create an Application Load Balancer using the AWS CLI, see [Tutorial: Create an Application Load Balancer using the AWS CLI](#) in the *User Guide for Application Load Balancers*.

This section walks you through the process of creating an Application Load Balancer in the AWS Management Console. For information about how to create an Application Load Balancer using the AWS CLI, see [Tutorial: Create an Application Load Balancer using the AWS CLI](#) in the *User Guide for Application Load Balancers*.

## Configure a target group for 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.

Each target group is used to route requests to one or more registered targets. When a rule condition is met, traffic is forwarded to the corresponding 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 create a target group

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation pane, under **Load Balancing**, choose **Target Groups**.
3. Choose **Create target group**.
4. For **Choose a target type**, select the target type.

#### Important

If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), you must choose **IP addresses** as the target type. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

5. For **Target group name**, type a name for the target group. This name must be unique per region per account, can have a maximum of 32 characters, must contain only alphanumeric characters or hyphens, and must not begin or end with a hyphen.
6. (Optional) For **Protocol** and **Port**, modify the default values as needed.
7. If the target type is **IP addresses**, choose **IPv4**.
8. For **VPC**, select a virtual private cloud (VPC). Note that for **IP addresses** target types, the VPCs available for selection are those that support the **IP address type** that you chose in the previous step.
9. (Optional) For **Protocol version**, leave the default.
10. (Optional) In the **Health checks** section, keep the default settings.
11. (Optional) Add one or more tags as follows:
  - a. Expand the **Tags** section.
  - b. Choose **Add tag**.
  - c. Enter the tag key and the tag value.
12. Choose **Next**.
13. Choose **Create target group**.

### 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 configure your load balancer and listener

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Load Balancing**, choose **Load Balancers**.
3. Choose **Create Load Balancer**.
4. Under **Application Load Balancer**, choose **Create**.
5. Under **Basic configuration**, do the following:
  - a. For **Load balancer name**, enter a name for your load balancer. For example, `my-alb`. The name of your Application Load Balancer must be unique within your set of Application Load Balancers

and Network Load Balancers for the Region. Names can have a maximum of 32 characters, and can contain only alphanumeric characters and hyphens. They cannot begin or end with a hyphen, or with `internal-`.

- b. For **Scheme**, choose **Internet-facing** or **Internal**. An internet-facing load balancer routes requests from clients to targets over the internet. An internal load balancer routes requests to targets using private IP addresses.
  - c. For **IP address type**, choose **IPv4** or **Dualstack**. Use **IPv4** if your clients use IPv4 addresses to communicate with the load balancer. Choose **Dualstack** if your clients use both IPv4 and IPv6 addresses to communicate with the load balancer.
6. Under **Network mapping**, do the following:
  - a. For **VPC**, select the same VPC that you used for the container instances on which you intend to run your service.
  - b. For **Mappings**, select the check box for the Availability Zones to use 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.
7. Under **Security groups**, do the following:

For **Security groups**, select an existing security group, or create a new one.

The security group for your load balancer must allow it to communicate with registered targets on both the listener port and the health check port. The console can create a security group for your load balancer on your behalf with rules that allow this communication. You can also create a security group and select it instead. For information about how to create a security group, see [Security groups for your Application Load Balancer](#) in *Elastic Load Balancing Application Load Balancers*.

(Optional) To create a new security group for your load balancer, choose **Create a new security group**.

8. Under **Listeners and routing**, do the following:

The default listener accepts HTTP traffic on port 80. You can keep the default protocol and port. For **Default action**, choose the target group that you created. You can optionally choose **Add listener** to add another listener (for example, an HTTPS listener).

If you create an HTTPS listener, configure the required

When you use HTTPS for your load balancer listener, you must deploy an SSL certificate on your load balancer. The load balancer uses this certificate to terminate the connection and decrypt requests from clients before sending them to the targets. Under **Secure listener settings**, do the following:

- a. For **Select policy**, choose a predefined security policy. For details on the security policies, see [Security Policies](#) in *Elastic Load Balancing Application Load Balancers*.
  - b. For **Default SSL certificate**, do one of the following:
    - If you created or imported a certificate using AWS Certificate Manager, select **From ACM**, and then select the certificate.
    - If you uploaded a certificate using IAM, select **From IAM**, and then select the certificate.
    - If you want to import a certificate to ACM or IAM, enter a certificate name. Then, paste the PEM-encoded private key and body.
9. (Optional) You can use **Add-on services**, such as the **AWS Global Accelerator** to create an accelerator and associate the load balancer with the accelerator. The accelerator name can have up to 64 characters. Allowed characters are a-z, A-Z, 0-9, . and - (hyphen). After the accelerator is created, you can use the **AWS Global Accelerator** console to manage it.
10. (Optional) Tag your Application Load Balancer. Under **Tag and create**, do the following

- a. Expand the **Tags** section.
  - b. Choose **Add tag**.
  - c. Enter the tag key and the tag value.
11. Review your configuration, and choose **Create 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 inbound rules**.
5. For **Type**, choose **All traffic**.
6. For **Source**, choose **Custom**, and then select the Application Load Balancer security group. 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 an Amazon ECS service](#) (p. 226).

## 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, H1, 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 an Amazon ECS service](#) (p. 226).

## 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. You must use [UpdateService](#) to modify the load balancer configuration of an existing service.

Multiple target groups can be specified in a service definition using the following format. For the full syntax of a service definition, see [Service definition template](#) (p. 224).

```
"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.

- For services that use an Application Load Balancer or Network Load Balancer, you cannot attach more than five target groups to a service.
- Specifying multiple target groups in a service definition is only supported under the following conditions:
  - The service must use either an Application Load Balancer or Network Load Balancer.
  - The service must use the rolling update (ECS) deployment controller type.
- Specifying multiple target groups is supported for services containing tasks using both the Fargate and EC2 launch types.
- When creating a service that specifies multiple target groups, the Amazon ECS service-linked role must be created. The role is created by omitting the `role` parameter in API requests, or the `Role` property in AWS CloudFormation. For more information, see [Service-linked role for Amazon ECS](#) (p. 348).

## 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. 224).

### 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. 293). 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. 265)—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. 269)—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.

## Service auto scaling and deployments

Application Auto Scaling disables scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. If you want to suspend scale-out processes while deployments are in progress, take the following steps.

1. Call the [describe-scalable-targets](#) command, specifying the resource ID of the ECS service associated with the scalable target in Application Auto Scaling (Example: `service/default/sample-webapp`). Record the output. You will need it when you call the next command.
2. Call the [register-scalable-target](#) command, specifying the resource ID, namespace, and scalable dimension. Specify `true` for both `DynamicScalingInSuspended` and `DynamicScalingOutSuspended`.
3. After deployment is complete, you can call the [register-scalable-target](#) command to resume scaling.

For more information, see [Suspending and resuming scaling for Application Auto Scaling](#).



## 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:DescribeAlarmsForMetric",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "cloudwatch:DisableAlarmActions",
        "cloudwatch:EnableAlarmActions",
        "iam:CreateServiceLinkedRole",
        "sns:CreateTopic",
        "sns:Subscribe",
        "sns:Get*",
        "sns:List*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The [Create Service Example \(p. 338\)](#) and [Update Service Example \(p. 339\)](#) 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. The `sns:` permissions are for the notifications that CloudWatch sends to an Amazon SNS topic when a threshold has been exceeded. If you use automatic scaling for your Amazon ECS services, it creates a service-linked role named `AWSServiceRoleForApplicationAutoScaling_ECSService`. This service-linked role grants Application Auto Scaling permission to describe the alarms for your policies, to monitor the current running task count of the service, and to modify the desired count of the service. The original managed Amazon ECS role for Application Auto Scaling was `ecsAutoscaleRole`, but it is no longer required. The service-linked role is the default role for Application Auto Scaling. For more information, see [Service-linked roles for Application Auto Scaling](#) in the *Application Auto Scaling User Guide*.

## 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.
- Application Auto Scaling disables scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. For more information, see [Service auto scaling and deployments \(p. 263\)](#).
- 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 you are not sure, you can create them by following the steps in [Set up to use Amazon ECS \(p. 4\)](#).

## 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 use 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.

## Service : sample-app-service

Cluster [service-autoscaling](#)

Status **ACTIVE**

Task definition [first-run-task-definition:5](#)

Launch type FARGATE

Platform version LATEST

Service role [aws-service-role/ecs.amazonaws.com/AWSServ](#)

Details

Tasks

Events

Auto Scaling

Deployments

### Load Balancing

Target Group Name	Container Name	Com
<a href="#">EC2Co-Defau-13FL25TVMRZRO</a>	sample-app	

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 \(p. 308\)](#) and [Updating cluster settings \(p. 83\)](#).

## 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 \(p. 265\)](#).

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

When using scaling policies, note the following considerations:

- Amazon ECS sends metrics in 1-minute intervals to CloudWatch. Metrics are not available until the clusters and services send the metrics to CloudWatch, and you cannot create CloudWatch alarms for metrics that do not exist yet.
- The scaling policies support a cooldown period. This is the number of seconds to wait for a previous scaling activity to take effect.
  - For scale-out events, the intention is to continuously (but not excessively) scale out. After Service Auto Scaling successfully scales out using a scaling policy, it starts to calculate the cooldown time. The scaling policy won't increase the desired capacity again unless either a larger scale out is triggered or the cooldown period ends. While the scale-out cooldown period is in effect, the capacity added by the initiating scale-out activity is calculated as part of the desired capacity for the next scale-out activity.
  - For scale-in events, the intention is to scale in conservatively to protect your application's availability, so scale-in activities are blocked until the cooldown period has expired. However, if another alarm triggers a scale-out activity during the scale-in cooldown period, Service Auto Scaling scales out the target immediately. In this case, the scale-in cooldown period stops and doesn't complete.
- The 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. This is what processing capacity is supposed to be. This prevents excessive (runaway) scaling that might not be satisfied, for example, if there aren't enough container instance resources to place the additional tasks. If the container instance capacity is available later, the pending scaling activity may succeed, and then further scaling activities can continue after the cooldown period.
- If you want your task count to scale to zero when there's no work to be done, set a minimum capacity of 0. With target tracking scaling policies, when actual capacity is 0 and the metric indicates that there is workload demand, Service Auto Scaling waits for one data point to be sent before scaling out. In this case, it scales out by the minimum possible amount as a starting point and then resumes scaling based on the actual running task count.
- Application Auto Scaling disables scale-in processes while Amazon ECS deployments are in progress. However, scale-out processes continue to occur, unless suspended, during a deployment. For more information, see [Service auto scaling and deployments \(p. 263\)](#).

## Amazon ECS console experience

Service Auto Scaling is off by default. You can turn it on 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 an Amazon ECS service \(p. 226\)](#) and [Updating a service \(p. 237\)](#). For a target tracking walkthrough, see [Target tracking scaling policies \(p. 265\)](#).

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 Amazon 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

Amazon ECS services can be configured to use 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*.

When you create a private namespace in AWS Cloud Map, we automatically create a Route 53 hosted zone. The Amazon ECS service can register with a friendly and predictable DNS name in Amazon Route 53. The hosted zones are updated automatically as your Amazon ECS services scale up or down. Other services perform a DNS lookup to determine whether to make a connection to the service.

Service discovery is available in the following AWS Regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1



Region Name	Region
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
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 (Milan)	eu-south-1
Europe (Stockholm)	eu-north-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1
AWS GovCloud (US-East)	us-gov-east-1
AWS GovCloud (US-West)	us-gov-west-1

## Service Discovery concepts

Service discovery consists of the following components:

- **Service discovery namespace:** A logical group of service discovery services that share the same domain name, such as `example.com`. This is the domain name where you want to route traffic to. You can create a namespace with a call to the `aws servicediscovery create-private-dns-namespace` command or in the Amazon ECS classic console. You can use the `aws servicediscovery list-namespaces` command to view the summary information about the namespaces that were created by the current account. For more information about the service discovery commands, see [create-private-dns-namespace](#) and [list-namespaces](#) in the *AWS Cloud Map (service discovery) AWS CLI Reference Guide*.
- **Service discovery service:** Exists within the service discovery namespace and consists of the service name and DNS configuration for the namespace. It provides the following core component:
  - **Service registry:** Allows you to look up a service via DNS or AWS Cloud Map API actions and get back one or more available endpoints that can be used to connect to the service.
- **Service discovery instance:** Exists within the service discovery service and consists of the attributes that are 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's 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 that's associated with the service discovery service.
- **AVAILABILITY\_ZONE** – The Availability Zone that the task was launched into. For tasks that use the EC2 launch type, this is the Availability Zone where the container instance is in. For tasks that use the Fargate launch type, this is the Availability Zone where the elastic network interface is in.
- **REGION** – The Region where the task is in.
- **ECS\_SERVICE\_NAME** – The name of the Amazon ECS service that the task belongs to.
- **ECS\_CLUSTER\_NAME** – The name of the Amazon ECS cluster that the task belongs to.
- **EC2\_INSTANCE\_ID** – The ID of the container instance the task was placed on. This custom attribute isn't 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 contains the external ID of the task set.
- **Amazon ECS health checks:** Amazon ECS performs periodic container-level health checks. If an endpoint doesn't pass the health check, it's removed from DNS routing and marked as unhealthy.

## Service discovery considerations

The following should be considered when using service discovery:

- Service discovery is supported for tasks on Fargate that use platform version 1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 58\)](#).
- Services configured to use service discovery have a limit of 1,000 tasks for each service. This is due to a Route 53 service quota.
- The Create Service workflow in the Amazon ECS classic console only supports registering services into private DNS namespaces. When an AWS Cloud Map private DNS namespace is created, a Route 53 private hosted zone will be created automatically.
- The VPC DNS attributes must be configured for successful DNS resolution. For information about how to configure the attributes, see [DNS support in your VPC](#) in the *Amazon VPC User Guide*.
- The DNS records created for a service discovery service always register with the private IP address for the task, rather than the public IP address, even when public namespaces are used.
- Service discovery requires that tasks specify either the `awsvpc`, `bridge`, or `host` network mode (none is not supported).
- If the service task definition uses the `awsvpc` network mode, you can create any combination of A or SRV records for each service task. If you use SRV records, a port is required.
- If the service task definition uses the `bridge` or `host` network mode, the SRV record is the only supported DNS record type. Create a SRV record for each service task. The SRV record must specify a container name and container port combination from the task definition.
- DNS records for a service discovery service can be queried within your VPC. They use the following format: `<service discovery service name>.<service discovery namespace>`.
- When doing a DNS query on the service name, A records return a set of IP addresses that correspond to your tasks. SRV records return a set of IP addresses and ports for each task.
- If you have eight or fewer healthy records, Route 53 responds to all DNS queries with all of the healthy records.
- When all records are unhealthy, Route 53 responds to DNS queries with up to eight unhealthy records.
- You can configure service discovery for an ECS service that's behind a load balancer, but service discovery traffic is always routed to the task and not the load balancer.

- Service discovery doesn't support the use of Classic Load Balancers.
- We recommend you use container-level health checks managed by Amazon ECS for your service discovery service.
  - **HealthCheckCustomConfig**—Amazon ECS manages health checks on your behalf. Amazon ECS uses information from container and health checks, and your task state, to update the health with AWS Cloud Map. This is specified using the `--health-check-custom-config` parameter when creating your service discovery service. For more information, see [HealthCheckCustomConfig](#) in the *AWS Cloud Map API Reference*.
- If you're using the Amazon ECS classic console, the workflow creates one service discovery service for each 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 isn't supported.
- The AWS Cloud Map resources created when service discovery is used must be cleaned up manually.

## Amazon ECS classic console experience

The workflow to create a service in the Amazon ECS classic 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 isn't supported.

To create a new Amazon ECS service that uses service discovery, see [Creating an Amazon ECS service](#) (p. 226).

## Service discovery pricing

Customers using Amazon ECS service discovery are charged for Route 53 resources and AWS Cloud Map discovery API operations. This involves costs for creating the Route 53 hosted zones and queries to the service registry. For more information, see [AWS Cloud Map Pricing](#) in the *AWS Cloud Map Developer Guide*.

Amazon ECS performs container level health checks and exposes them to AWS Cloud Map custom health check API operations. This is currently made available to customers at no extra cost. If you configure additional network health checks for publicly exposed tasks, you're charged for those health checks.

## 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 a `PENDING` to a `STOPPED` status), 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. This behavior reduces the effect that unstartable tasks have on your Amazon ECS cluster resources or Fargate infrastructure costs. If your service initiates the throttle logic, you receive the following [service event message](#) (p. 452):

```
(service service-name) is unable to consistently start tasks successfully.
```

Amazon ECS doesn't ever stop a failing service from retrying. It also doesn't attempt to modify it in any way other than increasing the time between restarts. The service throttle logic doesn't 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 \(p. 237\)](#).

The following are some common causes that trigger this logic:

- The Amazon ECS container agent can't pull your task Docker image. This might be because 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 \(p. 442\)](#).

### **Important**

Tasks that are stopped after they reach the `RUNNING` state don't start the throttle logic or the associated service event message. For example, assume that failed Elastic Load Balancing health checks for a service cause a task to be flagged as unhealthy, and Amazon ECS deregisters it and stops the task. At this point, the tasks aren't throttled. Even if a task's container command immediately exits with a non-zero exit code, the task already moved to the `RUNNING` state. Tasks that fail immediately because command errors don't cause 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 provide an overview on these resources and tags and show how you can use them.

## Contents

- [Tagging your Amazon ECS resources \(p. 276\)](#)
- [Amazon ECS service quotas \(p. 281\)](#)
- [Supported Regions for Amazon ECS on AWS Fargate \(p. 285\)](#)
- [Amazon ECS usage reports \(p. 287\)](#)

## Tagging your Amazon ECS resources

To help you manage your Amazon ECS resources, you can optionally assign your own metadata to each resource using *tags*. This topic provides an overview of tags in Amazon ECS and how you can create them.

To use this feature, you must opt in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 197\)](#).

### Important

Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags are not intended to be used for private or sensitive data.

## 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 can define a set of tags for your account's Amazon ECS container instances to help you track each 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 earlier 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 use AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to manage tags.

## Tagging your resources

You can tag new or existing Amazon ECS tasks, services, task definitions, and clusters.

### Important

Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags are not intended to be used for private or sensitive data.

If you're using the Amazon ECS console, you can apply tags to new or existing resources by using the **Tags** tab on the relevant resource page at any time. When you use the Amazon ECS-managed tags option (the **Propagate tags from** option), tags are copied from the task definition or service to a task. This can be done when you're running a task or creating a 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, alternatively, you can use the `TagResource` API action to apply tags to existing resources. For more information, see [TagResource](#). The `propagateTags` parameter can be used to copy tags from the task definition or service to the task. This can be done when you're running a task or creating a 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 can't be applied while resources are being created, we roll back the process of creating resources. This ensures that resources are either created with tags or not created at all, and that no resources are left untagged at any time. By tagging resources while they're being created, you can eliminate the need to run custom tagging scripts after resource creation.

When you use the ECS managed tags option, Amazon ECS automatically propagates tags to your tasks from either your task definition or your service.

The following table describes the Amazon ECS resources that can be tagged, and the resources that can be tagged when created.

### 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

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (Amazon ECS API, AWS CLI, AWS SDK)
Amazon ECS External instances	Yes	No	No, you can add tags after the External instance has been registered to a cluster using the AWS Management Console or by using the Amazon ECS API, AWS CLI, or AWS SDK. For more information, see <a href="#">Adding and deleting tags on an individual resource using the classic console (p. 279)</a> .
Amazon ECS capacity provider	Yes. Predefined <code>FARGATE</code> and <code>FARGATE_SPOT</code> capacity providers cannot be tagged.	No	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. These are reserved only 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 you use Amazon ECS-managed tags, Amazon ECS automatically tags all newly launched tasks with the cluster name. For tasks that belong to a service, they are also tagged with the service name. These managed tags are helpful when reviewing cost allocation after enabling them in your Cost and Usage Report. For more information, see [Amazon ECS usage reports \(p. 287\)](#).

To see the cost of your combined resources, you can organize your billing information based on resources that have the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several

services. For more information about setting up a cost allocation report with tags, see [The Monthly Cost Allocation Report](#) in the *AWS Billing User Guide*.

**Important**

To use this feature, you must opt in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 197\)](#).

**Note**

If you've 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) that supports tags, you can view and manage its tags on the **Tags** tab.

**Important**

Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags are not intended to be used for private or sensitive data.

**Contents**

- [Adding tags on an individual resource during launch \(p. 279\)](#)
- [Adding and deleting tags on an individual resource using the classic console \(p. 279\)](#)

## Adding tags on an individual resource during launch

The following resources enable you to specify tags when you create the resource.

Task	Console
Run one or more tasks.	<a href="#">Run a standalone task (p. 202)</a>
Create a service.	<a href="#">Creating an Amazon ECS service (p. 226)</a>
Create a task set.	<a href="#">External deployment (p. 247)</a>
Register a task definition.	<a href="#">Creating a task definition using the new console (p. 66)</a>
Create a cluster.	<a href="#">Creating a cluster using the classic console (p. 77)</a>

## Adding and deleting tags on an individual resource using the classic console

Amazon ECS enables you to add or delete tags associated with your clusters, services, tasks, and task definitions directly from the resource's page.



### 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**.

## 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.

### Important

Do not add personally identifiable information (PII) or other confidential or sensitive information in tags. Tags are accessible to many AWS services, including billing. Tags are not intended to be used for private or sensitive data.

### Tagging support for Amazon ECS resources

Task	AWS CLI	API action
Add or overwrite one or more tags.	<a href="#">tag-resource</a>	<a href="#">TagResource</a>
Delete one or more tags.	<a href="#">untag-resource</a>	<a href="#">UntagResource</a>

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.	<a href="#">run-task</a>	<a href="#">Start-ECSTask</a>	<a href="#">RunTask</a>
Create a service.	<a href="#">create-service</a>	<a href="#">New-ECSService</a>	<a href="#">CreateService</a>
Create a task set.	<a href="#">create-task-set</a>	<a href="#">New-ECSTaskSet</a>	<a href="#">CreateTaskSet</a>
Register a task definition.	<a href="#">register-task-definition</a>	<a href="#">Register-ECSTaskDefinition</a>	<a href="#">RegisterTaskDefinition</a>
Create a cluster.	<a href="#">create-cluster</a>	<a href="#">New-ECSCluster</a>	<a href="#">CreateCluster</a>

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

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 service quotas

The following tables provide 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.* For information on API throttling in the Amazon ECS API, see [Request throttling for the Amazon ECS API](#).

## Amazon ECS service quotas

The following are Amazon ECS service quotas.

Name	Default	Adjust	Description
Capacity providers per cluster	Each supported Region: 10	No	The maximum number of capacity providers that can be associated with a cluster.
Classic Load Balancers per service	Each supported Region: 1	No	The maximum number of Classic Load Balancers per service.
Clusters per account	Each supported Region: 10,000	Yes	Number of clusters per account
Container instances per cluster	Each supported Region: 5,000	No	Number of container instances per cluster
Container instances per start-task	Each supported Region: 10	No	The maximum number of container instances specified in a StartTask API action.
Containers per task definition	Each supported Region: 10	No	The maximum number of containers definitions within a task definition.
ECS Exec sessions	Each supported Region: 20	Yes	The maximum number of ECS Exec sessions per container.
Rate of tasks launched by a service on AWS Fargate	Each supported Region: 500	Yes	The maximum number of tasks that can be provisioned per service per minute on Fargate by the Amazon ECS service scheduler.
Rate of tasks launched by a service on an Amazon EC2 or External instance	Each supported Region: 500	Yes	The maximum number of tasks that can be provisioned per service per minute on an Amazon EC2 or External instance by the Amazon ECS service scheduler.
Revisions per task definition family	Each supported Region: 1,000,000	No	The maximum number of revisions per task definition family. Deregistering a task definition revision does not exclude it from being included in this limit.

Name	Default	Adjust	Description
Security groups per awsvpcConfiguration	Each supported Region: 5	No	The maximum number of security groups specified within an awsvpcConfiguration.
Services per cluster	Each supported Region: 5,000	Yes	The maximum number of services per cluster
Subnets per awsvpcConfiguration	Each supported Region: 16	No	The maximum number of subnets specified within an awsvpcConfiguration.
Tags per resource	Each supported Region: 50	No	The maximum number of tags per resource. This applies to task definitions, clusters, tasks, and services.
Target groups per service	Each supported Region: 5	No	The maximum number of target groups per service, if using an Application Load Balancer or a Network Load Balancer.
Task definition size	Each supported Region: 64 Kilobytes	No	The maximum size, in KiB, of a task definition.
Tasks in PROVISIONING state per cluster	Each supported Region: 300	No	The maximum number of tasks waiting in the PROVISIONING state per cluster. This quota only applies to tasks launched using an EC2 Auto Scaling group capacity provider.
Tasks launched per run-task	Each supported Region: 10	No	The maximum number of tasks that can be launched per RunTask API action.
Tasks per service	Each supported Region: 5,000	Yes	The maximum number of tasks per service (the desired count).

#### Note

Services configured to use Amazon ECS service discovery have a limit of 1,000 tasks per service. This is due to the AWS Cloud Map service quota for the number of instances per service. For more information, see [AWS Cloud Map service quotas](#) in the *Amazon Web Services General Reference*.

#### Note

In practice, task launch rates are also dependent on other considerations such as container images to be downloaded and unpacked, health checks and other integrations enabled, such as registering tasks with a load balancer. You will see variations in task launch rates compared with the quotas represented above based on the features that you have enabled for your Amazon ECS services. For more information, see [speeding up Amazon ECS deployments](#) in the Amazon ECS Best Practices Guide.

## AWS Fargate service quotas

The following are Amazon ECS on AWS Fargate service quotas.

Name	Default	Adjust	Description
Fargate On-Demand resource count	Each supported Region: 1,000	Yes	The maximum number of Amazon ECS tasks and Amazon EKS pods running concurrently on Fargate in this account in the current Region.
Fargate Spot resource count	Each supported Region: 1,000	Yes	The maximum number of Amazon ECS tasks running concurrently on Fargate Spot in this account in the current Region.

### Note

Fargate additionally enforces Amazon ECS tasks and Amazon EKS pods launch rate limits. For more information, see [Fargate throttling limits](#).

## Managing your Amazon ECS and AWS Fargate service quotas in the AWS Management Console

Amazon ECS has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see [What Is Service Quotas?](#) in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of your Amazon ECS service quotas.

AWS Management Console

### To view Amazon ECS and Fargate service quotas using the AWS Management Console

1. Open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Elastic Container Service (Amazon ECS)** or **AWS Fargate**.

In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.

4. To view additional information about a service quota, such as the description, choose the quota name.
5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the AWS Management Console see the [Service Quotas User Guide](#). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

AWS CLI

### To view Amazon ECS and Fargate service quotas using the AWS CLI

Run the following command to view the default Amazon ECS quotas.

```
aws service-quotas list-aws-default-service-quotas \
  --query 'Quotas[*]'.
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
  --service-code ecs \
  --output table
```

Run the following command to view the default Fargate quotas.

```
aws service-quotas list-aws-default-service-quotas \
  --query 'Quotas[*]'.
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
  --service-code fargate \
  --output table
```

Run the following command to view your applied Fargate quotas.

```
aws service-quotas list-service-quotas \
  --service-code fargate
```

**Note**

Amazon ECS does not support applied quotas.

To work more with service quotas using the AWS CLI, see the [Service Quotas AWS CLI Command Reference](#). To request a quota increase, see the [request-service-quota-increase](#) command in the [AWS CLI Command Reference](#).

## Supported Regions for Amazon ECS on AWS Fargate

**Contents**

- [Supported Regions for Linux containers on AWS Fargate \(p. 285\)](#)
- [Supported Regions for Windows containers on AWS Fargate \(p. 286\)](#)

### Supported Regions for Linux containers on AWS Fargate

Amazon ECS Linux containers 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 (usw1-az1 & usw1-az3 only)
US West (Oregon)	us-west-2

Region Name	Region
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Tokyo)	ap-northeast-1 (apne1-az1, apne1-az2, & apne1-az4 only)
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Canada (Central)	ca-central-1 (cac1-az1 & cac1-az2 only)
China (Beijing)	cn-north-1 (cnn1-az1 & cnn1-az2 only)
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 (Milan)	eu-south-1
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

## Supported Regions for Windows containers on AWS Fargate

Amazon ECS Windows containers 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 (use1-az1, use1-az2, use1-az4, use1-az5, & use1-az6only)

Region Name	Region
US West (N. California)	us-west-1 (usw1-az1 & usw1-az3 only)
US West (Oregon)	us-west-2 us-west-2 (usw2-az1, usw2-az2, & usw2-az3 only)
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2 ap-northeast-2 (apne2-az1, & apne2-az3 only)
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1 (apne1-az1, apne1-az2, & apne1-az4 only)
Canada (Central)	ca-central-1 (cac1-az1 & cac1-az2 only)
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Milan)	eu-south-1
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
Middle East (Bahrain)	me-south-1

## 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. For example, you can use it to identify areas that need further inquiry and see trends that you can use to understand your costs. You also can specify time ranges for the data, and view time data by day or by month.

The metering data in your Cost and Usage Report shows usage across all of your Amazon ECS tasks. The metering data includes CPU usage as `vCPU-Hours` and memory usage as `GB-Hours` for each task that was run. How that data is presented depends on the launch type of the task.

For tasks using the Fargate launch type, the `lineItem/Operation` column shows `FargateTask` and you will see the cost associated with each task.



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 \(p. 278\)](#).

**Important**

The metering data is only viewable for tasks launched on or after November 16, 2018. Tasks launched before this date don't show metering data.

Here's an example of some of the fields you can sort cost allocation data by 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 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?

When you are using the Fargate launch type, you get CPU and memory utilization metrics for each of your services to assist in the monitoring of your environment.

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.

## Topics

- [Monitoring tools \(p. 289\)](#)
- [Amazon ECS CloudWatch metrics \(p. 290\)](#)
- [AWS Fargate usage metrics \(p. 296\)](#)
- [Amazon ECS events and EventBridge \(p. 297\)](#)
- [Amazon ECS CloudWatch Container Insights \(p. 308\)](#)
- [Collecting application trace data \(p. 310\)](#)
- [Collecting application metrics \(p. 312\)](#)
- [Logging Amazon ECS API calls with AWS CloudTrail \(p. 317\)](#)

## 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 \(p. 290\)](#).

For services with tasks that use the Fargate launch type, you can use CloudWatch alarms to scale in and scale out the tasks in your service based on CloudWatch metrics, such as CPU and memory utilization. For more information, see [Service auto scaling \(p. 263\)](#).

- Amazon CloudWatch Logs – Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the `awslogs` log driver in your task definitions. For more information, see [Using the awslogs log driver \(p. 138\)](#).
- 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 \(p. 297\)](#) 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 \(p. 317\)](#) 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](#).

Amazon ECS collects metrics for clusters and services. You must enable Amazon ECS CloudWatch Container Insights for per-task metrics, including CPU and memory utilization. For more information about Container Insights, see [Amazon ECS CloudWatch Container Insights \(p. 308\)](#).

#### Topics

- [Enabling CloudWatch metrics \(p. 291\)](#)
- [Available metrics and dimensions \(p. 291\)](#)
- [Service utilization \(p. 293\)](#)
- [Service RUNNING task count \(p. 294\)](#)
- [Viewing Amazon ECS metrics \(p. 295\)](#)

## 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.

## 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, Amazon ECS provides CPU, memory, and GPU reservation and CPU and memory utilization metrics at the cluster and service level. You need to monitor the Amazon EC2 instances that make your underlying infrastructure separately.

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.

## 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 the Fargate launch type.

$$\text{Service CPU utilization} = \frac{(\text{Total CPU units used by tasks in service}) \times 100}{(\text{Total CPU units specified in task definition}) \times (\text{number of tasks in service})}$$

$$\text{Service memory utilization} = \frac{100 \times (\text{Total MiB of memory used by tasks in service})}{(\text{Total MiB of memory specified in task definition}) \times (\text{number of tasks in service})}$$

Each minute, the Amazon ECS container agent associated with each task calculates the number of CPU units and MiB of memory that are currently being used for each task owned by the service, 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](#).

**Note**

In this example, the CPU utilization will only go above 100% when the CPU units are defined at the container level. If you define CPU units at the task level, the utilization will not go above the defined task-level limit.

## Service `RUNNING` task count

You can use CloudWatch metrics to view the number of tasks in your services that are in the `RUNNING` state. For example, you can set a CloudWatch alarm for this metric to alert you if the number of running tasks in your service falls below a specified value.

## Service `RUNNING` task count in Amazon ECS CloudWatch Container Insights

A "Number of Running Tasks" (`RunningTaskCount`) metric is available per cluster and per service when you use Amazon ECS CloudWatch Container Insights. You can use Container Insights for all new clusters created by opting in to the `containerInsights` account setting, on individual clusters by enabling it using the cluster settings during cluster creation, or on existing clusters by using the `UpdateClusterSettings` API. Metrics collected by CloudWatch Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see [CloudWatch Pricing](#).

To view this metric, see [Amazon ECS Container Insights Metrics](#) in the *Amazon CloudWatch User Guide*.

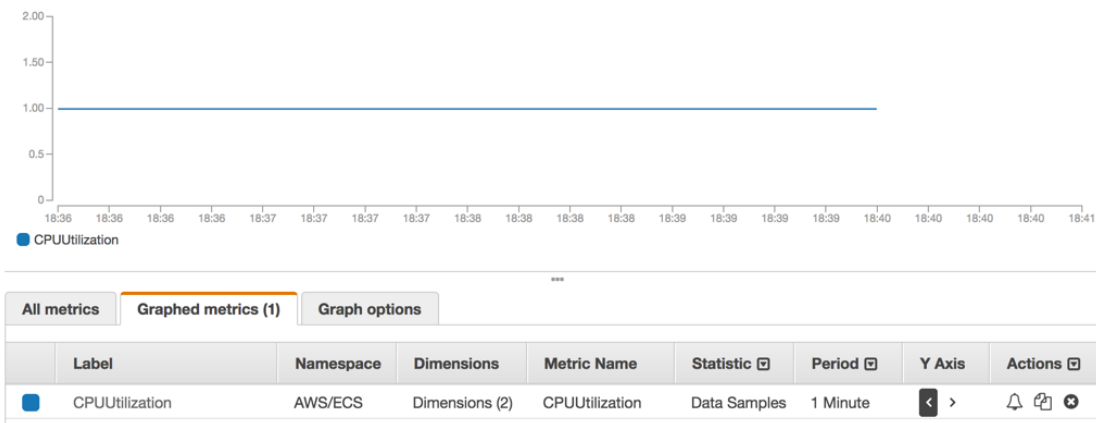
## Service `RUNNING` task count from Amazon ECS provided metrics

However Amazon ECS provides monitoring metrics at no additional cost. To use these metrics to count the running tasks, follow the steps below in the CloudWatch console.

### 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, All metrics**.
3. On the **Browse** tab, choose the **ECS** namespace.
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 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 service metrics using the Amazon ECS console \(p. 295\)](#)
- [Viewing Amazon ECS metrics using the CloudWatch console \(p. 296\)](#)

## Viewing service metrics using 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 \(p. 293\)](#).

### New console

1. Open the new console at <https://console.aws.amazon.com/ecs/v2>.
2. Select the cluster that you want to view metrics for.
3. On the **Cluster: *cluster-name*** page, select the service.

The metrics are available under **Health**.

### Classic 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**.

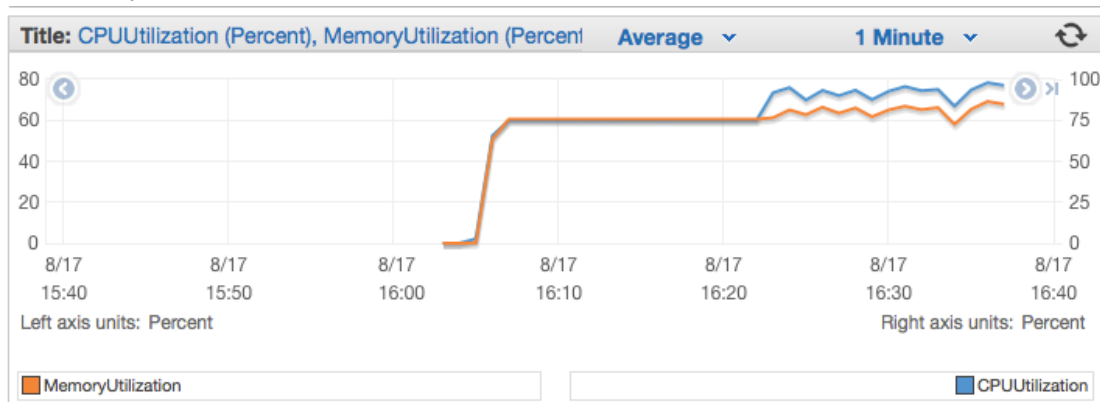


## Viewing Amazon ECS metrics using the CloudWatch console

Amazon ECS service metrics can also be viewed on the CloudWatch console. The console provides the most detailed view of Amazon ECS metrics, and you can tailor the views to suit your needs. You can view [Service utilization](#) (p. 293) and the [Service RUNNING task count](#) (p. 294). For more information about CloudWatch, 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 **All Metrics, 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.



## AWS Fargate usage metrics

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

AWS Fargate usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about Fargate service quotas, see [AWS Fargate service quotas](#) (p. 284).

AWS Fargate publishes the following metrics in the `AWS/Usage` namespace.

Metric	Description
<code>ResourceCount</code>	The total number of the specified resource running on your account. The resource is defined by the dimensions associated with the metric.

The following dimensions are used to refine the usage metrics that are published by AWS Fargate.

Dimension	Description
<code>Service</code>	The name of the AWS service containing the resource. For AWS Fargate usage metrics, the value for this dimension is <code>Fargate</code> .

Dimension	Description
Type	The type of entity that is being reported. Currently, the only valid value for AWS Fargate usage metrics is <code>Resource</code> .
Resource	<p>The type of resource that is running.</p> <p>Currently, AWS Fargate returns information on your Fargate On-Demand and Fargate Spot usage. The resource value for Fargate On-Demand usage is <code>OnDemand</code> and for Fargate Spot it is <code>Spot</code>.</p> <p><b>Note</b> Fargate On-Demand usage combines Amazon EKS pods using Fargate, Amazon ECS tasks using the Fargate launch type and Amazon ECS tasks using the <code>FARGATE</code> capacity provider.</p>
Class	The class of resource being tracked. Currently, AWS Fargate does not use the class dimension.

## Creating a CloudWatch alarm to monitor Fargate resource usage metrics

AWS Fargate provides CloudWatch usage metrics that correspond to the AWS service quotas for Fargate On-Demand and Fargate Spot resource usage. In the Service Quotas console, you can visualize your usage on a graph and configure alarms that alert you when your usage approaches a service quota. For more information, see [AWS Fargate usage metrics \(p. 296\)](#).

Use the following steps to create a CloudWatch alarm based on one of the Fargate resource usage metrics.

### To create an alarm based on your Fargate usage 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 **AWS Fargate**.
4. In the **Service quotas** list, select the Fargate usage quota you want to create an alarm for.
5. In the Amazon CloudWatch Events alarms section, choose **Create**.
6. For **Alarm threshold**, choose the percentage of your applied quota value that you want to set as the alarm value.
7. For **Alarm name**, enter a name for the alarm and then choose **Create**.

## 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. When using the Fargate launch type, you can see the state of your tasks. 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. 306).

Events are relatively ordered, so that you can easily tell when an event occurred in relation to other events.

#### Topics

- [Amazon ECS events](#) (p. 298)
- [Handling events](#) (p. 306)

## Amazon ECS events

Amazon ECS tracks the state of each of your tasks and services. If the state of a task or service changes, an event is triggered and is sent to Amazon EventBridge. These events are classified as task state change events and service action events. These events and their possible causes are described in greater detail in the following sections.

#### 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.

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 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.

## 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 your tasks and it reports any state changes. State changes might include changes from `PENDING` to `RUNNING` or from `RUNNING` to `STOPPED`.

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-8ac87ddb5b8",
```

```

        "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"
  }
}
```

## 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. The service scheduler reports the status periodically, so you might receive this message multiple times.

#### **TASKSET\_STEADY\_STATE**

The task set is healthy and at the desired number of tasks, thus reaching a steady state.

#### **CAPACITY\_PROVIDER\_STEADY\_STATE**

A capacity provider associated with a service reaches a steady state.

#### **SERVICE\_DESIRED\_COUNT\_UPDATED**

When the service scheduler updates the computed desired count for a service or task set. This event is not sent when the desired count is manually updated by a user.

### 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 \(p. 197\)](#). 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*.

```
{
  "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"
    }
  }
}

```

In the following example for the EC2 launch type, the task was attempted to launch on the Container Instance `2dd1b186f39845a584488d2ef155c131` but the service scheduler was unable to place the task because of insufficient CPU.

```

{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3a6d0468b",
  "detail-type": "ECS Service Action",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2019-11-19T19:55:38Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "ERROR",
    "eventName": "SERVICE_TASK_PLACEMENT_FAILURE",
    "clusterArn": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
    "containerInstanceArns": [
      "arn:aws:ecs:us-west-2:111122223333:container-instance/default/2dd1b186f39845a584488d2ef155c131"
    ],
    "reason": "RESOURCE:CPU",
    "createdAt": "2019-11-06T19:09:33.087Z"
  }
}

```

## Service deployment state change events

Amazon ECS sends service deployment change state events with the detail type **ECS Deployment State Change**. The following is an event pattern that is used to create an EventBridge rule for Amazon ECS service deployment state change events. For more information, see [Creating an EventBridge Rule](#) in the *Amazon EventBridge User Guide*.

```

{
  "source": [
    "aws.ecs"
  ],
  "detail-type": [
    "ECS Deployment State Change"
  ]
}

```

Amazon ECS sends events with **INFO** and **ERROR** event types. The following are the service deployment state change events.

#### SERVICE\_DEPLOYMENT\_IN\_PROGRESS

The service deployment is in progress. This event is sent for both initial deployments and rollback deployments.

#### SERVICE\_DEPLOYMENT\_COMPLETED

The service deployment has completed. This event is sent once a service reaches a steady state after a deployment.

#### SERVICE\_DEPLOYMENT\_FAILED

The service deployment has failed. This event is sent for services with deployment circuit breaker logic enabled.

### Example service deployment in progress event

Service deployment in progress events are delivered when both an initial and a rollback deployment is started. The difference between the two is in the `reason` field. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

The following shows an example output for an initial deployment starting.

```
{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3a6EXAMPLE",
  "detail-type": "ECS Deployment State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2020-05-23T12:31:14Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "INFO",
    "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
    "deploymentId": "ecs-svc/123",
    "updatedAt": "2020-05-23T11:11:11Z",
    "reason": "ECS deployment deploymentId in progress."
  }
}
```

The following shows an example output for a rollback deployment starting. The `reason` field provides the ID of the deployment the service is rolling back to.

```
{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3a6EXAMPLE",
  "detail-type": "ECS Deployment State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2020-05-23T12:31:14Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "INFO",
    "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
    "deploymentId": "ecs-svc/123",
    "updatedAt": "2020-05-23T11:11:11Z",

```

```

    "reason": "ECS deployment circuit breaker: rolling back to
deploymentId deploymentID."
  }
}

```

### Example service deployment completed event

Service deployment completed state events are delivered in the following format. For more information, see [Rolling update \(p. 240\)](#).

```

{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
  "detail-type": "ECS Deployment State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2020-05-23T12:31:14Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "INFO",
    "eventName": "SERVICE_DEPLOYMENT_COMPLETED",
    "deploymentId": "ecs-svc/123",
    "updatedAt": "2020-05-23T11:11:11Z",
    "reason": "ECS deployment deploymentID completed."
  }
}

```

### Example service deployment failed event

Service deployment failed state events are delivered in the following format. A service deployment failed state event will only be sent for services that have deployment circuit breaker logic enabled. For more information, see [Rolling update \(p. 240\)](#).

```

{
  "version": "0",
  "id": "ddca6449-b258-46c0-8653-e0e3aEXAMPLE",
  "detail-type": "ECS Deployment State Change",
  "source": "aws.ecs",
  "account": "111122223333",
  "time": "2020-05-23T12:31:14Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
  ],
  "detail": {
    "eventType": "ERROR",
    "eventName": "SERVICE_DEPLOYMENT_FAILED",
    "deploymentId": "ecs-svc/123",
    "updatedAt": "2020-05-23T11:11:11Z",
    "reason": "ECS deployment circuit breaker: task failed to start."
  }
}

```

## Handling 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 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 task state change events and saves them to the following Amazon DynamoDB table:

- *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"]
    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, service, and task level as CloudWatch metrics. For more information, see [Amazon ECS Container Insights metrics](#) 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. 290).

## 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 available for all tasks run on Fargate and tasks run on Amazon EC2 instances that use either the `bridge` or `awsvpc` network modes.

## Setting up CloudWatch Container Insights for cluster and service level metrics

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 using the classic console](#) (p. 77).

### Important

#### 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
```

## Collecting application trace data

Amazon ECS integrates with AWS Distro for OpenTelemetry to collect trace data from your application. Amazon ECS uses an AWS Distro for OpenTelemetry sidecar container to collect and route trace data to AWS X-Ray. For more information, see [Setting up AWS Distro for OpenTelemetry Collector in Amazon ECS](#).

For the AWS Distro for OpenTelemetry Collector to send trace data to AWS X-Ray, your application must be configured to create the trace data. For more information, see [Instrumenting your application for AWS X-Ray](#) in the *AWS X-Ray Developer Guide*.

## Required IAM permissions for AWS Distro for OpenTelemetry integration with AWS X-Ray

The Amazon ECS integration with AWS Distro for OpenTelemetry requires that you create a task IAM role and specify the role in your task definition. We recommend that the AWS Distro for OpenTelemetry sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The new Amazon ECS console experience takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually. For more information about creating a task execution IAM role, see [Amazon ECS task execution IAM role](#) (p. 354).

### Important

If you're also collecting application metrics using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see [Collecting application metrics](#) (p. 312).

### To create a task IAM role for AWS Distro for OpenTelemetry integration

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies, Create policy**.
3. On the **Create policy** page, switch to the **JSON** tab, copy and paste the following IAM policy JSON into the field, then choose **Next: Tags**.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",
```

```
        "Action": [
            "xray:PutTraceSegments",
            "xray:PutTelemetryRecords",
            "xray:GetSamplingRules",
            "xray:GetSamplingTargets",
            "xray:GetSamplingStatisticSummaries"
        ],
        "Resource": "*"
    }
}
```

4. (Optional) Add one or more tags to the policy, then choose **Next: Review**.
5. For **Name**, specify `AWSDistroOpenTelemetryPolicyForXray`.
6. For **Description**, specify an optional description, then choose **Create policy**.
7. In the navigation pane, choose **Roles, Create role**.
8. In the **Select type of trusted entity** section, choose **AWS service, Elastic Container Service**.
9. For **Select your use case**, choose **Elastic Container Service Task**, then choose **Next: Permissions**.
10. In the **Attach permissions policy** section, search for `AWSDistroOpenTelemetryPolicyForXray`, select the policy, and then choose **Next: Tags**.
11. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
12. For **Role name**, specify `AmazonECS_OpenTelemetryXrayRole` and choose **Create role**.

## Specifying the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration in your task definition

The new Amazon ECS console experience simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use trace collection** option. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition. The following task definition snippet shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for AWS X-Ray integration.

```
{
  "family": "otel-using-xray",
  "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryXrayRole",
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
  "containerDefinitions": [{
    "name": "aws-otel-emitter",
    "image": "application-image",
    "logConfiguration": {
      "logDriver": "awslogs",
      "options": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/aws-otel-emitter",
        "awslogs-region": "us-east-1",
        "awslogs-stream-prefix": "ecs"
      }
    }
  }],
  "dependsOn": [{
    "containerName": "aws-otel-collector",
    "condition": "START"
  }]
}
```



```
"name": "aws-otel-collector",
"image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.17.0",
"essential": true,
"command": [
  "--config=/etc/ecs/ecs-xray.yaml"
],
"logConfiguration": {
  "logDriver": "awslogs",
  "options": {
    "awslogs-create-group": "True",
    "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "ecs"
  }
}
},
"networkMode": "awsvpc",
"requiresCompatibilities": [
  "FARGATE"
],
"cpu": "1024",
"memory": "3072"
}
```

## Collecting application metrics

The AWS Distro for OpenTelemetry (ADOT) metrics collection feature is in preview. The preview is open to all AWS accounts. Features may be added or changed before announcing General Availability.

Amazon ECS on Fargate supports collecting metrics from your applications running on Fargate and exporting them to either Amazon CloudWatch or Amazon Managed Service for Prometheus. Amazon ECS uses an AWS Distro for OpenTelemetry sidecar container to collect and route your application metrics to the destination. The new Amazon ECS console experience simplifies the process of adding this integration when creating your task definitions.

### Topics

- [Exporting application metrics to Amazon CloudWatch \(p. 312\)](#)
- [Exporting application metrics to Amazon Managed Service for Prometheus \(p. 315\)](#)

## Exporting application metrics to Amazon CloudWatch

The AWS Distro for OpenTelemetry (ADOT) metrics collection feature is in preview. The preview is open to all AWS accounts. Features may be added or changed before announcing General Availability.

Amazon ECS on Fargate supports exporting your custom application metrics to Amazon CloudWatch as custom metrics. This is done by adding the AWS Distro for OpenTelemetry sidecar container to your task definition. The new Amazon ECS console experience simplifies this process by adding the **Use metric collection** option when creating a new task definition. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

The application metrics are exported to CloudWatch Logs with log group name `/aws/ecs/application/metrics` and the metrics can be viewed in the `ECS/AWSOTel/Application` namespace. Your application must be instrumented with the OpenTelemetry SDK. For more information, see [Introduction to AWS Distro for OpenTelemetry](#) in the AWS Distro for OpenTelemetry documentation.

## Considerations

The following should be considered when using the Amazon ECS on Fargate integration with AWS Distro for OpenTelemetry to send application metrics to Amazon CloudWatch.

- When this integration is used, Amazon ECS doesn't send any task-level metrics to CloudWatch Container Insights. You can enable Container Insights at the Amazon ECS cluster level to receive those metrics. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 308\)](#).
- The AWS Distro for OpenTelemetry integration is only supported for Amazon ECS workloads hosted on Fargate. Amazon ECS workloads hosted on Amazon EC2 instances or external instances aren't currently supported.
- CloudWatch supports a maximum of 10 dimensions per metric. By default, Amazon ECS defaults to including the `TaskARN`, `ClusterARN`, `LaunchType`, `TaskDefinitionFamily`, and `TaskDefinitionRevision` dimensions to the metrics. The remaining 5 dimensions can be defined by your application. If more than 10 dimensions are configured, CloudWatch can't display them. When this occurs, the application metrics will appear in the `ECS/AWSOTel/Application` CloudWatch metric namespace but without any dimensions. You can instrument your application to add additional dimensions. For more information, see [Using CloudWatch metrics with AWS Distro for OpenTelemetry](#) in the AWS Distro for OpenTelemetry documentation.

## Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon CloudWatch

The Amazon ECS integration with AWS Distro for OpenTelemetry requires that you create a task IAM role and specify the role in your task definition. We recommend that the AWS Distro for OpenTelemetry sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The new Amazon ECS console experience takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually and added to your task definition. For more information about the task execution IAM role, see [Amazon ECS task execution IAM role \(p. 354\)](#).

### Important

If you're also collecting application trace data using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see [Collecting application trace data \(p. 310\)](#).

### To create a task IAM role for AWS Distro for OpenTelemetry integration with CloudWatch

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies, Create policy**.
3. On the **Create policy** page, switch to the **JSON** tab, copy and paste the following IAM policy JSON into the field, then choose **Next: Tags**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups",
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*"
    }
  ]
}
```

```
}  
]  
}
```

#### Note

If your application requires any additional permissions, you should add them to this policy. Each task definition may only specify one task IAM role. For example, if you are using a custom configuration file stored in Systems Manager, you should add the `ssm:GetParameters` permission to this IAM policy.

4. (Optional) Add one or more tags to the policy, then choose **Next: Review**.
5. For **Name**, specify `AWSDistroOpenTelemetryPolicyForCloudWatch`.
6. For **Description**, specify an optional description, then choose **Create policy**.
7. In the navigation pane, choose **Roles, Create role**.
8. In the **Select type of trusted entity** section, choose **AWS service, Elastic Container Service**.
9. For **Select your use case**, choose **Elastic Container Service Task**, then choose **Next: Permissions**.
10. In the **Attach permissions policy** section, search for **AWSDistroOpenTelemetryPolicyForCloudWatch**, select the policy, and then choose **Next: Tags**.
11. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
12. For **Role name**, specify `AmazonECS_OpenTelemetryCloudWatchRole` and choose **Create role**.

## Specifying the AWS Distro for OpenTelemetry sidecar in your task definition

The new Amazon ECS console experience simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use metric collection** option. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition manually. The following task definition example shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for Amazon CloudWatch integration.

```
{  
  "family": "otel-using-cloudwatch",  
  "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",  
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",  
  "containerDefinitions": [{  
    "name": "aws-otel-emitter",  
    "image": "application-image",  
    "logConfiguration": {  
      "logDriver": "awslogs",  
      "options": {  
        "awslogs-create-group": "true",  
        "awslogs-group": "/ecs/aws-otel-emitter",  
        "awslogs-region": "us-east-1",  
        "awslogs-stream-prefix": "ecs"  
      }  
    },  
    "dependsOn": [{  
      "containerName": "aws-otel-collector",  
      "condition": "START"  
    }]  
  }],  
  {  
    "name": "aws-otel-collector",  
    "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.17.0",  
  }  
}
```

```
"essential": true,
"command": [
  "--config=/etc/ecs/ecs-cloudwatch.yaml"
],
"logConfiguration": {
  "logDriver": "awslogs",
  "options": {
    "awslogs-create-group": "True",
    "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "ecs"
  }
}
},
"networkMode": "awsvpc",
"requiresCompatibilities": [
  "FARGATE"
],
"cpu": "1024",
"memory": "3072"
}
```

## Exporting application metrics to Amazon Managed Service for Prometheus

The AWS Distro for OpenTelemetry (ADOT) metrics collection feature is in preview. The preview is open to all AWS accounts. Features may be added or changed before announcing General Availability.

Amazon ECS supports exporting your task-level CPU, memory, network, and storage metrics and your custom application metrics to Amazon Managed Service for Prometheus. This is done by adding the AWS Distro for OpenTelemetry sidecar container to your task definition. The new Amazon ECS console experience simplifies this process by adding the **Use metric collection** option when creating a new task definition. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

The metrics are exported to Amazon Managed Service for Prometheus and can be viewed using the Amazon Managed Grafana dashboard. Your application must be instrumented with either Prometheus libraries or with the OpenTelemetry SDK. For more information about instrumenting your application with the OpenTelemetry SDK, see [Introduction to AWS Distro for OpenTelemetry](#) in the AWS Distro for OpenTelemetry documentation.

When using the Prometheus libraries, your application must expose a `/metrics` endpoint which is used to scrape the metrics data. For more information about instrumenting your application with Prometheus libraries, see [Prometheus client libraries](#) in the Prometheus documentation.

## Considerations

The following should be considered when using the Amazon ECS on Fargate integration with AWS Distro for OpenTelemetry to send application metrics to Amazon CloudWatch.

- The AWS Distro for OpenTelemetry integration is only supported for Amazon ECS workloads hosted on Fargate. Amazon ECS workloads hosted on Amazon EC2 instances or external instances aren't supported currently.
- Amazon ECS doesn't send any task-level metrics to CloudWatch Container Insights. You can enable Container Insights at the Amazon ECS cluster level to receive those metrics. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 308\)](#).
- By default, AWS Distro for OpenTelemetry includes all available task-level dimensions for your application metrics when exporting to Amazon Managed Service for Prometheus. You can also

instrument your application to add additional dimensions. For more information, see [Getting Started with Prometheus Remote Write Exporter for Amazon Managed Service for Prometheus](#) in the AWS Distro for OpenTelemetry documentation.

## Required IAM permissions for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus

The Amazon ECS integration with Amazon Managed Service for Prometheus using the AWS Distro for OpenTelemetry sidecar requires that you create a task IAM role and specify the role in your task definition. This task IAM role must be created manually using the steps below prior to registering your task definition.

We recommend that the AWS Distro for OpenTelemetry sidecar also be configured to route container logs to CloudWatch Logs which requires a task execution IAM role be created and specified in your task definition as well. The new Amazon ECS console experience takes care of the task execution IAM role on your behalf, but the task IAM role must be created manually. For more information about creating a task execution IAM role, see [Amazon ECS task execution IAM role \(p. 354\)](#).

### Important

If you're also collecting application trace data using the AWS Distro for OpenTelemetry integration, ensure your task IAM role also contains the permissions necessary for that integration. For more information, see [Collecting application trace data \(p. 310\)](#).

### To create a task IAM role for AWS Distro for OpenTelemetry integration with Amazon Managed Service for Prometheus

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 **AWS service**, **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 the **AmazonPrometheusRemoteWriteAccess** policy, select the policy, and then choose **Next: Tags**.
6. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
7. For **Role name**, specify `AmazonECS_OpenTelemetryPrometheusRole` and choose **Create role**.

## Specifying the AWS Distro for OpenTelemetry sidecar in your task definition

The new Amazon ECS console experience simplifies the experience of creating the AWS Distro for OpenTelemetry sidecar container by using the **Use metric collection** option. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

If you're not using the Amazon ECS console, you can add the AWS Distro for OpenTelemetry sidecar container to your task definition manually. The following task definition example shows the container definition for adding the AWS Distro for OpenTelemetry sidecar for Amazon Managed Service for Prometheus integration.

```
{
  "family": "otel-using-cloudwatch",
  "taskRoleArn": "arn:aws:iam::111122223333:role/AmazonECS_OpenTelemetryCloudWatchRole",
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
  "containerDefinitions": [{
```

```
"name": "aws-otel-emitter",
"image": "application-image",
"logConfiguration": {
  "logDriver": "awslogs",
  "options": {
    "awslogs-create-group": "true",
    "awslogs-group": "/ecs/aws-otel-emitter",
    "awslogs-region": "us-east-1",
    "awslogs-stream-prefix": "ecs"
  }
},
"dependsOn": [{
  "containerName": "aws-otel-collector",
  "condition": "START"
}]
},
{
  "name": "aws-otel-collector",
  "image": "public.ecr.aws/aws-observability/aws-otel-collector:v0.17.0",
  "essential": true,
  "command": [
    "--config=/etc/ecs/ecs-cloudwatch.yaml"
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "True",
      "awslogs-group": "/ecs/ecs-aws-otel-sidecar-collector",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
],
"networkMode": "awsvpc",
"requiresCompatibilities": [
  "FARGATE"
],
"cpu": "1024",
"memory": "3072"
}
```

## 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 \(p. 320\)](#)
- [Logging and Monitoring in Amazon Elastic Container Service \(p. 375\)](#)
- [Compliance Validation for Amazon Elastic Container Service \(p. 376\)](#)
- [Infrastructure Security in Amazon Elastic Container Service \(p. 376\)](#)

## 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 \(p. 321\)](#)
- [Authenticating with identities \(p. 321\)](#)
- [Managing access using policies \(p. 323\)](#)
- [How Amazon Elastic Container Service works with IAM \(p. 325\)](#)
- [Identity-based policy examples for Amazon Elastic Container Service \(p. 331\)](#)
- [AWS managed policies for Amazon Elastic Container Service \(p. 340\)](#)

- [Service-linked role for Amazon ECS \(p. 348\)](#)
- [Amazon ECS task execution IAM role \(p. 354\)](#)
- [ECS Anywhere IAM role \(p. 358\)](#)
- [IAM roles for tasks \(p. 360\)](#)
- [Amazon ECS CodeDeploy IAM Role \(p. 365\)](#)
- [Amazon ECS CloudWatch Events IAM Role \(p. 368\)](#)
- [Additional configuration for Windows IAM roles for tasks \(p. 371\)](#)
- [Troubleshooting Amazon Elastic Container Service identity and access \(p. 373\)](#)

## Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon ECS.

**Service user** – If you use the Amazon ECS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon ECS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon ECS, see [Troubleshooting Amazon Elastic Container Service identity and access \(p. 373\)](#).

**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 \(p. 325\)](#).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon ECS. To view example Amazon ECS identity-based policies that you can use in IAM, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 331\)](#).

## 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 [Signing in to the AWS Management Console as an IAM user or root user](#) 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 address or your IAM user name. You can access AWS programmatically using your root user or IAM users 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*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*.
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [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 or IAM users, 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. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

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, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [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. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

## Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

## Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of 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, learn what IAM features are available to use with Amazon ECS.

### IAM features you can use with Amazon Elastic Container Service

IAM feature	Amazon ECS support
<a href="#">Identity-based policies (p. 325)</a>	Yes
<a href="#">Resource-based policies (p. 326)</a>	No
<a href="#">Policy actions (p. 326)</a>	Yes
<a href="#">Policy resources (p. 327)</a>	Partial
<a href="#">Policy condition keys (p. 328)</a>	Yes
<a href="#">ACLs (p. 329)</a>	No
<a href="#">ABAC (tags in policies) (p. 330)</a>	Yes
<a href="#">Temporary credentials (p. 330)</a>	Yes
<a href="#">Principal permissions (p. 330)</a>	Yes
<a href="#">Service roles (p. 331)</a>	Yes
<a href="#">Service-linked roles (p. 331)</a>	Yes

To get a high-level view of how Amazon ECS and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

## Identity-based policies for Amazon ECS

Supports identity-based policies	Yes
----------------------------------	-----

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

### Identity-based policy examples for Amazon ECS

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 331\)](#).

## Resource-based policies within Amazon ECS

Supports resource-based policies	No
----------------------------------	----

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

## Policy actions for Amazon ECS

Supports policy actions	Yes
-------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon ECS actions, see [Actions defined by Amazon Elastic Container Service](#) in the *Service Authorization Reference*.

Policy actions in Amazon ECS use the following prefix before the action:

```
ecs
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
    "ecs:action1",  
    "ecs:action2"  
]
```

You can specify multiple actions using wildcards (\*). For example, to specify all actions that begin with the word `Describe`, include the following action:



```
"Action": "ecs:Describe*"
```

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service](#) (p. 331).

## Policy resources for Amazon ECS

Supports policy resources	Partial
---------------------------	---------

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (\*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon ECS resource types and their ARNs, see [Resources defined by Amazon Elastic Container Service](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions defined by Amazon Elastic Container Service](#).

Some Amazon ECS API actions support multiple resources. For example, multiple clusters can be referenced when calling the DescribeClusters API action. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
    "EXAMPLE-RESOURCE-1",  
    "EXAMPLE-RESOURCE-2"
```

For example, the Amazon ECS cluster resource has the following ARN:

```
arn:${Partition}:ecs:${Region}:${Account}:cluster/${clusterName}
```

To specify my-cluster-1 and my-cluster-2 cluster in your statement, use the following ARNs:

```
"Resource": [  
    "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-1",  
    "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster-2"
```

To specify all clusters that belong to a specific account, use the wildcard (\*):

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

For task definitions, you can specify the latest revision, or a specific revision.

To specify the latest task definition, use:



```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}"
```

To specify a specific task definition revision, use `${TaskDefinitionRevisionNumber}`:

```
"Resource:arn:${Partition}:ecs:${Region}:${Account}:task-definition/
${TaskDefinitionFamilyName}:${TaskDefinitionRevisionNumber}"
```

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service \(p. 331\)](#).

## Policy condition keys for Amazon ECS

Supports service-specific policy condition keys	Yes
---	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or *Condition block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

Amazon ECS supports the following service-specific condition keys that you can use to provide fine-grained filtering for your IAM policies:

Condition Key	Description	Evaluation Types
<code>aws:RequestTag/\${TagKey}</code>	<p>The context key is formatted <code>"aws:RequestTag/<i>tag-key</i>": "<i>tag-value</i>"</code> where <i>tag-key</i> and <i>tag-value</i> are a tag key and value pair.</p> <p>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".</p>	String
<code>aws:ResourceTag/\${TagKey}</code>	<p>The context key is formatted <code>"aws:ResourceTag/<i>tag-key</i>": "<i>tag-value</i>"</code> where <i>tag-key</i> and <i>tag-value</i> are a tag key and value pair.</p> <p>Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.</p>	String

Condition Key	Description	Evaluation Types
aws:TagKeys	This context key is formatted "aws:TagKeys": " <b>tag-key</b> " where <b>tag-key</b> 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/ <b>tag-key</b> ": " <b>tag-value</b> " where <b>tag-key</b> and <b>tag-value</b> 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": " <b>cluster-arn</b> " where <b>cluster-arn</b> is the ARN for the Amazon ECS cluster.	ARN, Null
ecs:container-instances	The context key is formatted "ecs:container-instances": " <b>container-instance-arns</b> " where <b>container-instance-arns</b> is one or more container instance ARNs.	ARN, Null
ecs:container-name	The context key is formatted "ecs:container-name": " <b>container-instance</b> " where <b>container-instance</b> is the name of an Amazon ECS container which is defined in the ECS task definition.	String
ecs:enable-execute-command	The context key is formatted "ecs:enable-execute-command": " <b>value</b> " where <b>value</b> is "true" or "false".	String
ecs:task-definition	The context key is formatted "ecs:task-definition": " <b>task-definition-arn</b> " where <b>task-definition-arn</b> is the ARN for the Amazon ECS task definition.	ARN, Null
ecs:service	The context key is formatted "ecs:service": " <b>service-arn</b> " where <b>service-arn</b> is the ARN for the Amazon ECS service.	ARN, Null

To see a list of Amazon ECS condition keys, see [Condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon Elastic Container Service](#).

To view examples of Amazon ECS identity-based policies, see [Identity-based policy examples for Amazon Elastic Container Service](#) (p. 331).

## Access control lists (ACLs) in Amazon ECS

Supports ACLs	No
---------------	----

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

## Attribute-based access control (ABAC) with Amazon ECS

Supports ABAC (tags in policies)	Yes
----------------------------------	-----

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

For more information about tagging Amazon ECS resources, see [Resources and tags \(p. 276\)](#).

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 \(p. 339\)](#).

## Using Temporary credentials with Amazon ECS

Supports temporary credentials	Yes
--------------------------------	-----

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

## Cross-service principal permissions for Amazon ECS

Supports principal permissions	Yes
--------------------------------	-----

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that

then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*.

## Service roles for Amazon ECS

Supports service roles	Yes
------------------------	-----

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

### Warning

Changing the permissions for a service role might break Amazon ECS functionality. Edit service roles only when Amazon ECS provides guidance to do so.

## Service-linked roles for Amazon ECS

Supports service-linked roles	Yes
-------------------------------	-----

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing Amazon ECS service-linked roles, see [Service-linked role for Amazon ECS](#) (p. 348).

## Identity-based policy examples for Amazon Elastic Container Service

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 actions on the resources that 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 IAM policies](#) in the *IAM User Guide*.

### Topics

- [Policy best practices](#) (p. 332)
- [Allow users to view their own permissions](#) (p. 332)
- [Amazon ECS first-run wizard permissions](#) (p. 333)
- [Cluster examples](#) (p. 336)
- [List and Describe Task Examples](#) (p. 338)
- [Create Service Example](#) (p. 338)
- [Update Service Example](#) (p. 339)
- [Describing Amazon ECS Services Based on Tags](#) (p. 339)

## 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:ListGroupsWithUser",
        "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 in the classic console 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 \(p. 341\)](#) 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 [Set up to use Amazon ECS \(p. 4\)](#).
- 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-linked role for Amazon ECS \(p. 348\)](#)
  - [Amazon ECS task execution IAM role \(p. 354\)](#)

## 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"
        }
      }
    }
  ]
}
```

```
}

```

## 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*.

## AWS managed policies for Amazon Elastic Container Service

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

Amazon ECS and Amazon ECR provide several managed policies and trust relationships that you can attach to AWS Identity and Access Management (IAM) users, Amazon EC2 instances, and Amazon ECS tasks that allow differing levels of control over resources and API operations. You can apply these policies directly, or you can use them as starting points for creating your own policies. For more information about the Amazon ECR managed policies, see [Amazon ECR managed policies](#).

## AmazonECS\_FullAccess

You can attach the `AmazonECS_FullAccess` policy to your IAM identities.

This policy grants administrative access to Amazon ECS resources and grants an IAM identity (such as a user, group, or role) access to the AWS services that Amazon ECS is integrated with to use all of Amazon ECS features. Using this policy allows access to all of Amazon ECS features that are available in the AWS Management Console.

### Permissions details

The `AmazonECS_FullAccess` managed IAM policy includes the following permissions. Following the best practice of granting least privilege, you can use the `AmazonECS_FullAccess` managed policy as a template for creating your own custom policy. That way, you can take away or add permissions to and from the managed policy based on your specific requirements.

- `ecs` – Allows principals full access to all Amazon ECS APIs.
- `application-autoscaling` – Allows principals to create, describe, and manage Application Auto Scaling resources. This is required when enabling service auto scaling for your Amazon ECS services.
- `appmesh` – Allows principals to list App Mesh service meshes and virtual nodes and describe App Mesh virtual nodes. This is required when integrating your Amazon ECS services with App Mesh.
- `autoscaling` – Allows principals to create, manage, and describe Amazon EC2 Auto Scaling resources. This is required when managing Amazon EC2 auto scaling groups when using the cluster auto scaling feature.
- `cloudformation` – Allows principals to create and manage AWS CloudFormation stacks. This is required when creating Amazon ECS clusters using the AWS Management Console and the subsequent managing of those clusters.
- `cloudwatch` – Allows principals to create, manage, and describe Amazon CloudWatch alarms.
- `codedeploy` – Allows principals to create and manage application deployments as well as view their configurations, revisions, and deployment targets.
- `sns` – Allows principals to view a list of Amazon SNS topics.
- `lambda` – Allows principals to view a list of AWS Lambda functions and their version specific configurations.
- `ec2` – Allows principals run Amazon EC2 instances as well as create and manage routes, route tables, internet gateways, launch groups, security groups, virtual private clouds, spot fleets, and subnets.
- `elasticloadbalancing` – Allows principals to create, describe, and delete Elastic Load Balancing load balancers. Principals will also be able to fully manage the target groups, listeners, and listener rules for load balancers.
- `events` – Allows principals to create, manage, and delete Amazon EventBridge rules and their targets.
- `iam` – Allows principals to list IAM roles and their attached policies. Principals can also list instance profiles available to your Amazon EC2 instances.
- `logs` – Allows principals to create and describe Amazon CloudWatch Logs log groups. Principals can also list log events for these log groups.
- `route53` – Allows principals to create, manage, and delete Amazon Route 53 hosted zones. Principals can also view Amazon Route 53 health check configuration and information. For more information about hosted zones, see [Working with hosted zones](#).
- `servicediscovery` – Allows principals to create, manage, and delete AWS Cloud Map services and create private DNS namespaces.

The following is an example `AmazonECS_FullAccess` policy.

```
{  
  "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"
      ]
    }
  }
}
```

## AmazonECSTaskExecutionRolePolicy

The AmazonECSTaskExecutionRolePolicy managed IAM policy grants the permissions that are needed by the Amazon ECS container agent and AWS Fargate container agents to make AWS API calls on your behalf. This policy can be added to your task execution IAM role. For more information, see [Amazon ECS task execution IAM role \(p. 354\)](#).

### Permissions details

The AmazonECSTaskExecutionRolePolicy managed IAM policy includes the following permissions. Following the standard security advice of granting least privilege, the AmazonECSTaskExecutionRolePolicy managed policy can be used as a guide. If any of the permissions that are granted in the managed policy aren't needed for your use case, create a custom policy and add only the permissions that you require.

- `ecr:GetAuthorizationToken` – Allows a principal to retrieve an authorization token. The authorization token represents your IAM authentication credentials and can be used to access any Amazon ECR registry that the IAM principal has access to. The authorization token received is valid for 12 hours.
- `ecr:BatchCheckLayerAvailability` – When a container image is pushed to an Amazon ECR private repository, each image layer is checked to verify if it's already pushed. If it's pushed, then the image layer is skipped.
- `ecr:GetDownloadUrlForLayer` – When a container image is pulled from an Amazon ECR private repository, this API is called once for each image layer that's not already cached.
- `ecr:BatchGetImage` – When a container image is pulled from an Amazon ECR private repository, this API is called once to retrieve the image manifest.
- `logs:CreateLogStream` – Allows a principal to create a CloudWatch Logs log stream for a specified log group.
- `logs:PutLogEvents` – Allows a principal to upload a batch of log events to a specified log stream.

The following is an example AmazonECSTaskExecutionRolePolicy policy.

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

## AWSApplicationAutoscalingECSServicePolicy

You can't attach `AWSApplicationAutoscalingECSServicePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Application Auto Scaling to perform actions on your behalf. For more information, see [Service-linked roles for Application Auto Scaling](#).

## AWSCodeDeployRoleForECS

You can't attach `AWSCodeDeployRoleForECS` to your IAM entities. This policy is attached to a service-linked role that allows CodeDeploy to perform actions on your behalf. For more information, see [Create a service role for CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

## AWSCodeDeployRoleForECSLimited

You can't attach `AWSCodeDeployRoleForECSLimited` to your IAM entities. This policy is attached to a service-linked role that allows CodeDeploy to perform actions on your behalf. For more information, see [Create a service role for CodeDeploy](#) in the *AWS CodeDeploy User Guide*.

## Amazon ECS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon ECS since this service started tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon ECS Document history page.

Change	Description	Date
Amazon ECS started tracking changes	Amazon ECS started tracking changes for its AWS managed policies.	June 8, 2021

## Phased out AWS managed IAM policies for Amazon Elastic Container Service

The following AWS managed IAM policies are phased out. These policies are now replaced by the updated policies. We recommend that you update your IAM users or roles to use the updated policies.

### AmazonEC2ContainerServiceFullAccess

#### Important

The `AmazonEC2ContainerServiceFullAccess` managed IAM policy was phased out as of January 29, 2021, in response to a security finding with the `iam:passRole` permission. This permission grants access to all resources including credentials to roles in the account. Now that the policy is phased out, you can't attach the policy to any new IAM users or roles. Any users or roles that already have the policy attached can continue using it. However, we recommend that you update your IAM users or roles to use the `AmazonECS_FullAccess` managed policy instead. For more information, see [Migrating to the AmazonECS\\_FullAccess managed policy](#) (p. 347).

### AmazonEC2ContainerServiceRole

#### Important

The `AmazonEC2ContainerServiceRole` managed IAM policy is phased out. It's now replaced by the Amazon ECS service-linked role. For more information, see [Service-linked role for Amazon ECS](#) (p. 348).

## AmazonEC2ContainerServiceAutoscaleRole

### Important

The AmazonEC2ContainerServiceAutoscaleRole managed IAM policy is phased out. It's now replaced by the Application Auto Scaling service-linked role for Amazon ECS. For more information, see [Service-linked roles for Application Auto Scaling](#) in the *Application Auto Scaling User Guide*.

## Migrating to the AmazonECS\_FullAccess managed policy

The AmazonEC2ContainerServiceFullAccess managed IAM policy was phased out on January 29, 2021, in response to a security finding with the `iam:passRole` permission. This permission grants access to all resources including credentials to roles in the account. Now that the policy is phased out, you can't attach the policy to any new IAM groups, users, or roles. Any groups, users, or roles that already have the policy attached can continue using it. However, we recommend that you update your IAM groups, users, or roles to use the AmazonECS\_FullAccess managed policy instead.

The permissions that are granted by the AmazonECS\_FullAccess policy include the complete list of permissions that are necessary to use ECS as an administrator. If you currently use permissions that are granted by the AmazonEC2ContainerServiceFullAccess policy that aren't in the AmazonECS\_FullAccess policy, you can add them to an in-line policy statement. For more information, see [AWS managed policies for Amazon Elastic Container Service \(p. 340\)](#).

Use the following steps to determine if you have any IAM groups, users, or roles that are currently using the AmazonEC2ContainerServiceFullAccess managed IAM policy. Then, update them to detach the earlier policy and attach the AmazonECS\_FullAccess policy.

### To update an IAM group, user, or role to use the AmazonECS\_FullAccess policy (AWS Management Console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies** and search for and select the AmazonEC2ContainerServiceFullAccess policy.
3. Choose the **Policy usage** tab that displays any IAM role that's currently using this policy.
4. For each IAM role that's currently using the AmazonEC2ContainerServiceFullAccess policy, select the role and use the following steps to detach the deprecated policy and attach the AmazonECS\_FullAccess policy.
  - a. On the **Permissions** tab, choose the **X** next to the AmazonEC2ContainerServiceFullAccess policy.
  - b. Choose **Add permissions**.
  - c. Choose **Attach existing policies directly**, search for and select the AmazonECS\_FullAccess policy, and then choose **Next: Review**.
  - d. Review the changes and then choose **Add permissions**.
  - e. Repeat these steps for each IAM group, user, or role that's using the AmazonEC2ContainerServiceFullAccess policy.

### To update an IAM group, user, or role to use the AmazonECS\_FullAccess policy (AWS CLI)

1. Use the [generate-service-last-accessed-details](#) command to generate a report that includes details about when the deprecated policy was last used.

```
aws iam generate-service-last-accessed-details \
  --arn arn:aws:iam::aws:policy/AmazonEC2ContainerServiceFullAccess
```

Example output:

```
{
  "JobId": "32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE"
}
```

2. Use the job ID from the previous output with the [get-service-last-accessed-details](#) command to retrieve the last accessed report of the service. This report displays the Amazon Resource Name (ARN) of the IAM entities that last used the deprecated policy.

```
aws iam get-service-last-accessed-details \
  --job-id 32bb1fb0-1ee0-b08e-3626-ae83EXAMPLE
```

3. Use one of the following commands to detach the `AmazonEC2ContainerServiceFullAccess` policy from an IAM group, user, or role.
  - [detach-group-policy](#)
  - [detach-role-policy](#)
  - [detach-user-policy](#)
4. Use one of the following commands to attach the `AmazonECS_FullAccess` policy to an IAM group, user, or role.
  - [attach-group-policy](#)
  - [attach-role-policy](#)
  - [attach-user-policy](#)

## Service-linked role for Amazon ECS

Amazon ECS uses a service-linked role for the permissions the service requires to call other AWS services on your behalf. For more information, see [Using service-linked roles](#) in the *IAM User Guide*.

### 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:*"
    },
    {
      "Sid": "ExecuteCommandSessionManagement",
      "Effect": "Allow",
      "Action": [
        "ssm:DescribeSessions"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ExecuteCommand",
      "Effect": "Allow",
      "Action": [
        "ssm:StartSession"
      ],
      "Resource": [
        "arn:aws:ecs:*:*:task/*",
        "arn:aws:ssm:*:*:document/AmazonECS-ExecuteInteractiveCommand"
      ]
    }
  ]
}

```

## 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

After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. You can't edit the AWS owned IAM policy that the Amazon ECS service-linked role uses either as it contains all the necessary permissions Amazon ECS needs. However, you can edit the description of the role.

### 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.

- Delete all Amazon ECS clusters in all regions. For more information, see [Deleting a cluster using the classic console \(p. 83\)](#).

## 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.

## Amazon ECS task execution IAM role

The task execution role grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your behalf. The task execution IAM role is required depending on the requirements of your task. You can have multiple task execution roles for different purposes and services associated with your account.

The following are common use cases for a task execution IAM role:

- Your task is hosted on AWS Fargate or on an external instance and...
  - is pulling a container image from an Amazon ECR private repository.
  - sends container logs to CloudWatch Logs using the `awslogs` log driver. For more information, see [Using the `awslogs` log driver](#) (p. 138).
- Your tasks are hosted on either AWS Fargate or Amazon EC2 instances and...
  - is using private registry authentication. For more information, see [Required IAM permissions for private registry authentication](#) (p. 356).
  - 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. 357).

**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 can be created for you in the Amazon ECS console; 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. 355). 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 **Add Permissions, 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**.
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, choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update 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 a task execution IAM role (AWS Management Console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**, **Create role**.
3. In the **Trusted entity type** section, choose **AWS service**, **Elastic Container Service**.
4. For **Use case**, choose **Elastic Container Service Task**, then choose **Next**.

5. In the **Attach permissions policy** section, do the following:
  - a. Search for **AmazonECSTaskExecutionRolePolicy**, then select the policy.
  - b. Under **Set permissions boundary - optional**, choose **Create role without a permissions boundary**.
  - c. Choose **Next**.
6. Under **Role details**, do the following:
  - a. For **Role name**, type `ecsTaskExecutionRole`.
  - b. For **Add tags (optional)**, specify any custom tags to associate with the policy .
7. Choose **Create role**.

### To create a task execution IAM role (AWS CLI)

1. Create a file named `ecs-tasks-trust-policy.json` that contains the trust policy to use for the IAM role. The file should contain the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Create an IAM role named `ecsTaskExecutionRole` using the trust policy created in the previous step.

```
aws iam create-role \
  --role-name ecsTaskExecutionRole \
  --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

3. Attach the AWS managed `AmazonECSTaskExecutionRolePolicy` policy to the `ecsTaskExecutionRole` role. This policy provides

```
aws iam attach-role-policy \
  --role-name ecsTaskExecutionRole \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

## 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 \(p. 174\)](#).

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",
        "ssm:GetParameters",
        "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 \(p. 176\)](#).

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-xxxxxx"
        }
      }
    }
  ]
}
```

## ECS Anywhere IAM role

When registering an on-premise server or virtual machine (VM) to your cluster, the server or VM requires an IAM role to communicate with AWS APIs. You only need to create this IAM role once per AWS account.

### To check for the `ecsAnywhereRole` IAM role (AWS Management 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 `ecsAnywhereRole`. 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. On the **Permissions** tab, ensure that the **AmazonEC2ContainerServiceforEC2Role** and **AmazonSSMManagedInstanceCore** managed policies are attached to the role. If the policy is attached, your Amazon ECS Anywhere role is properly configured. If not, follow the steps below to attach the policies.
  - a. Choose **Attach policies**.
  - 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**.
  - d. In the **Filter** box, type **AmazonSSMManagedInstanceCore** to narrow the available policies to attach.
  - e. Check the box to the left of the **AmazonSSMManagedInstanceCore** policy and choose **Attach policy**.
5. Choose the **Trust relationships** tab, and **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": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ssm.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

### To create the `ecsAnywhereRole` IAM role (AWS Management Console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Choose the **AWS service** role type, and then choose **Elastic Container Service**.
4. Choose the **EC2 Role for Elastic Container Service** use case and then **Next: Permissions**.
5. In the **Attached permissions policy** section, select **AmazonEC2ContainerServiceforEC2Role** and then choose **Next: Review**.
6. For **Role name**, type `ecsAnywhereRole` and optionally you can enter a description, for example **Allows on-premises servers or virtual machine in an ECS cluster to access ECS..**
7. Review your role information and then choose **Create role** to finish.
8. Choose the `ecsAnywhereRole` role you just created.
9. On the **Permissions** tab, choose **Attach policies**.
10. In the **Filter** box, type **AmazonSSMManagedInstanceCore** to narrow the available policies to attach.



11. Check the box to the left of the **AmazonSSMManagedInstanceCore** policy and choose **Attach policy**.
12. On the **Trust relationships** tab, choose **Edit trust relationship**.
13. Change the trust relationship so that it contains the following policy and then choose **Update Trust Policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ssm.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

### To create the `ecsAnywhereRole` IAM role (AWS CLI)

1. Create a local file named `ssm-trust-policy.json` with the following contents.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": [
      "ssm.amazonaws.com"
    ]},
    "Action": "sts:AssumeRole"
  }
}
```

2. Create the role.

```
aws iam create-role --role-name ecsAnywhereRole --assume-role-policy-document
file://ssm-trust-policy.json
```

3. Attach the AWS managed policies.

```
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam attach-role-policy --role-name ecsAnywhereRole --policy-arn
arn:aws:iam::aws:policy/service-role/AmazonEC2ContainerServiceforEC2Role
```

## IAM roles for tasks

Your Amazon ECS tasks can have an IAM role associated with them. The permissions granted in the IAM role are assumed by the containers running in the task.

If your containerized applications need to call AWS APIs, they must sign their AWS API requests with AWS credentials, and a task IAM role provides a strategy for managing credentials for your applications to use, similar to the way that an Amazon EC2 instance profile provides credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you can associate an IAM role with an Amazon ECS task definition or `RunTask` API

operation. Your containers can then use the AWS SDK or AWS CLI to make API requests to authorized AWS services.

The following explain the benefits of using IAM roles with your 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.

#### 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 Amazon EC2 or external instance they are running on.

You can specify a task IAM role 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`.

From inside the container, you can query the credential endpoint 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"
}
```

## Creating an IAM role and policy for your tasks

When creating an IAM policy for your tasks to use, the policy should include the permissions that you would like the containers in your tasks to assume. You can use an existing AWS managed policy that as an example or you can create a custom policy from scratch that meets your specific needs. For more information, see [Creating IAM policies](#) in the *IAM User Guide*.

#### Important

When creating your task IAM role, it is recommended that you use the `aws:SourceAccount` or `aws:SourceArn` condition keys in either the trust relationship or the IAM policy associated with the role to prevent the confused deputy security issue. These condition keys can be specified in the trust relationship or in the IAM policy associated with the role. To learn more about the

confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

Once the IAM policy is created, you can create an IAM role which includes that policy which you reference in your Amazon ECS task definition. You can create the role using the **Elastic Container Service Task** use case in the IAM console. Then you can attach your specific IAM policy to the role that gives the containers in your task the permissions you desire. The procedures below describe how to do this.

If you have multiple task definitions or services that require IAM permissions, you should consider creating a role for each specific task definition or service with the minimum required permissions for the tasks to operate so that you can minimize the access that you provide for each task.

For information about the service endpoint for your Region, see [Service endpoints](#) in the *Amazon Web Services General Reference Reference Guide*.

The IAM task role must have a trust policy that specifies the `ecs-tasks.amazonaws.com` service. The `sts:AssumeRole` permission allows your tasks to assume an IAM role that's different from the one that the Amazon EC2 instance uses. This way, your task doesn't inherit the role associated with the Amazon EC2 instance. It is recommended that you use the `aws:SourceAccount` or `aws:SourceArn` condition keys to scope the permissions further to prevent the confused deputy security issue. These condition keys can be specified in the trust relationship or in the IAM policy associated with the role. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

The following is an example trust policy. You should replace the Region identifier and specify the AWS account number that you use when launching tasks.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ecs-tasks.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:ecs:us-west-2:111122223333:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

### To create an IAM policy for your tasks (AWS Management Console)

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.
4. (Optional) For **Request conditions**, select **Add condition**. This is recommended to prevent the confused deputy security issue. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.
  - a. For **Condition key**, select either **aws:SourceAccount** or **aws:SourceArn**. For more information about these global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.
  - b. For **Operator**, select **StringEquals** if you specified the **aws:SourceAccount** condition key or **ArnLike** if you specified the **aws:SourceArn** condition key.
  - c. For **Value**, specify your AWS account ID if you specified the **aws:SourceAccount** condition key or the Amazon Resource Name (ARN) of your Amazon ECS task if you specified the **aws:SourceArn** condition key. You may use wildcards, for example `aws:ecs:*:accountId:*` which will work for all tasks in your account.
  - d. Choose **Add** to save the condition key. Repeat these steps for each condition key you want to add to the policy.
5. Choose **Next: Tags** and add any resource tags to the policy to help you organize them and then choose **Next: Review**.
6. On the **Review policy** page, for **Name** type your own unique name, such as `AmazonECSTaskS3BucketPolicy`. You may specify an optional description for the policy as well.
7. When the policy is complete, 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. It includes a condition statement, which you can use to specify either a specific task using its Amazon Resource Name (ARN) or a specific account ID. This provides a way to further scope the permission for additional security. This is recommended to prevent the confused deputy security issue. To learn more about the confused deputy problem and how to protect your AWS account, see [The confused deputy problem](#) in the *IAM User Guide*.

The following is an example permissions policy. You can modify the policy to suit your specific needs. You should replace the Region identifier and specify the AWS account number that you use when launching tasks.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3::my-task-secrets-bucket/*"
      ],
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:ecs:us-west-2:111122223333:*"
        }
      }
    }
  ]
}
```

```
    },  
    "StringEquals": {  
      "aws:SourceAccount": "111122223333"  
    }  
  }  
}  
]  
}
```

2. Choose **Next: Tags** and add any resource tags to the policy to help you organize them and then choose **Next: Review**.
3. On the **Review policy** page, for **Name** type your own unique name, such as `AmazonECSTaskS3BucketPolicy`. You may specify an optional description for the policy as well.
4. When the policy is complete, choose **Create policy** to finish.

### To create an IAM role for your tasks (AWS Management Console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select trusted entity** section, choose **AWS service**.
4. For **Use case**, using the drop down menu, select **Elastic Container Service** and then the **Elastic Container Service Task** use case and then choose **Next**.
5. For **Add permissions**, search for and select the policy to use for your tasks (in this example `AmazonECSTaskS3BucketPolicy`, and then choose **Next**.
6. On **Step 3: Name, review, and create**, do the following:
  - a. For **Role name**, enter a name for your role. For this example, type `AmazonECSTaskS3BucketRole` to name the role.
  - b. (Optional) For **Description**, specify a description for this IAM role.
  - c. Review the trusted entity and permissions policy for the role.
  - d. For **Add tags (Optional)**, enter any metadata tags you want to associate with the IAM role, and then choose **Create role**.

## 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 classic console to create your task definition, choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify the Amazon Resource Name (ARN) of your task role using the `taskRoleArn` parameter. For more information, see [Creating a task definition using the new console \(p. 66\)](#).

**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 classic 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 [Run a standalone task \(p. 202\)](#).

**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 \(p. 246\)](#).

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:GetObjectVersion"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iam:PassedToService": [
            "ecs-tasks.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

```
    }
  ]
}
```

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:GetObjectVersion"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
        }
      },
      "Effect": "Allow"
    },
    {
      "Action": [
        "iam:PassRole"
      ],

```

```

    "Effect": "Allow",
    "Resource": [
      "arn:aws:iam::*:role/ecsTaskExecutionRole",
      "arn:aws:iam::*:role/ECSTaskExecution*"
    ],
    "Condition": {
      "StringLike": {
        "iam:PassedToService": [
          "ecs-tasks.amazonaws.com"
        ]
      }
    }
  }
]
}

```

### 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**.
6. In the **Attach permissions policy** section, do the following:
  - a. Search for **AWSCodeDeployRoleForECS**, then select the policy.
  - b. Under **Set permissions boundary - optional**, choose **Create role without a permissions boundary**.
  - c. Choose **Next**.
7. Under **Role details**, do the following:
  - a. For **Role name**, type `ecsCodeDeployRole`, and enter an optional description.
  - b. For **Add tags (optional)**, specify any custom tags to associate with the policy .
8. 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 **Add Permissions, 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**.
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, choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update 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 \(p. 354\)](#) and [IAM roles for tasks \(p. 360\)](#).

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 \(p. 204\)](#).

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 \(p. 354\)](#).

#### 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. In the navigation pane, choose **Roles**.
4. 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.

5. On the **Permissions** tab, ensure that the **AmazonEC2ContainerServiceEventsRole** 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 **Add Permissions, Attach policies**.
  - b. To narrow the available policies to attach, for **Filter**, type **AmazonEC2ContainerServiceEventsRole**.
  - c. Check the box to the left of the **AmazonEC2ContainerServiceEventsRole** policy and choose **Attach policy**.
6. Choose **Trust relationships**.
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, choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update 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, Create role**.
3. In the **Trusted entity type** section, choose **AWS service, Elastic Container Service**.
4. For **Use case**, choose **Elastic Container Service Task**, then choose **Next**.
5. In the **Attach permissions policy** section, do the following:
  - a. Search for **AmazonEC2ContainerServiceEventsRole**, then select the policy.
  - b. Under **Set permissions boundary - optional**, choose **Create role without a permissions boundary**.
  - c. Choose **Next**.
6. Under **Role details**, do the following:
  - a. For **Role name**, type `ecsEventsRole`.
  - b. For **Add tags (optional)**, specify any custom tags to associate with the policy.
7. Choose **Create role**.
8. Search the list of roles for `ecsEventsRole` and select the role you just created.
9. On the **Permissions** tab, choose **Add Permissions, Attach policies**.
10. Replace the existing trust relationship with the following text. Choose **Edit trust policy**, copy the policy into the **Policy Document** window and choose **Update 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**.

## Additional configuration for Windows IAM roles for tasks

The IAM roles for tasks with Windows features requires additional configuration, but much of this configuration is similar to configuring IAM roles for tasks on Linux container instances. The following requirements must be met to configure IAM roles for tasks for Windows containers.

- When you launch your container instances, you must set the `-EnableTaskIAMRole` option in the container instances user data script. The `EnableTaskIAMRole` turns on the Task IAM roles feature for the tasks. For example:

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster 'windows' -EnableTaskIAMRole
</powershell>
```

- You must bootstrap your container with the networking commands that are provided in [IAM roles for task container bootstrap script](#) (p. 372).
- 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. 361).
- Your container must use an AWS SDK that supports IAM roles for tasks. For more information, see [Using a supported AWS SDK](#) (p. 364).
- 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. 364).
- The IAM roles for the task credential provider use port 80 on the container instance. Therefore, if you configure IAM roles for tasks on your container instance, your containers can't use port 80 for the host port in any port mappings. To expose your containers on port 80, we recommend configuring a service for them that uses load balancing. You can use port 80 on the load balancer. By doing so, traffic can be routed to another host port on your container instances. For more information, see [Service load balancing](#) (p. 252).
- 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.

### Note

You do not need to run this script when you use `awsvpc` network mode on Windows.

If you run Windows containers which include Powershell, then use the following script:

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

$gateway = (Get-NetRoute | Where { $_.DestinationPrefix -eq '0.0.0.0/0' } | Sort-Object
RouteMetric | Select NextHop).NextHop
$ifIndex = (Get-NetAdapter -InterfaceDescription "Hyper-V Virtual Ethernet*" | Sort-Object
| Select ifIndex).ifIndex
New-NetRoute -DestinationPrefix 169.254.170.2/32 -InterfaceIndex $ifIndex -NextHop $gateway
-PolicyStore ActiveStore # credentials API
New-NetRoute -DestinationPrefix 169.254.169.254/32 -InterfaceIndex $ifIndex -NextHop
$gateway -PolicyStore ActiveStore # metadata API
```

If you run Windows containers that only have the Command shell, then use the following script:

```
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
```

```
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

for /f "tokens=1" %i in ('netsh interface ipv4 show interfaces ^| findstr /x /r
    ".*vEthernet.*"') do set interface=%i
for /f "tokens=3" %i in ('netsh interface ipv4 show addresses %interface% ^| findstr /x /r
    ".*Default.Gateway.*"') do set gateway=%i
netsh interface ipv4 add route prefix=169.254.170.2/32 interface="%interface%"
    nexthop="%gateway%" store=active # credentials API
netsh interface ipv4 add route prefix=169.254.169.254/32 interface="%interface%"
    nexthop="%gateway%" store=active # metadata API
```

## 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 \(p. 373\)](#)
- [I am not authorized to perform iam:PassRole \(p. 373\)](#)
- [I want to view my access keys \(p. 374\)](#)
- [I'm an administrator and want to allow others to access Amazon ECS \(p. 374\)](#)
- [I want to allow people outside of my AWS account to access my Amazon ECS resources \(p. 374\)](#)

## 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 fictional `my-example-widget` resource but does not have the fictional `ecs:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
ecs:GetWidget on resource: my-example-widget
```

In this case, 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 \(p. 325\)](#).
- 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. 290).

For services with tasks that use the Fargate launch type, you can use CloudWatch alarms to scale in and scale out the tasks in your service based on CloudWatch metrics, such as CPU and memory utilization. For more information, see [Service auto scaling](#) (p. 263).

### Amazon CloudWatch Logs

Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the `awslogs` log driver in your task definitions. For more information, see [Using the awslogs log driver](#) (p. 138).

### 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. 297) 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. 317).

### 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. Amazon ECS supports resource-based access policies, which can include restrictions based on the source IP address, so make sure that the policies account for the IP address for the network location. You can also use Amazon ECS policies to control access from specific Amazon Virtual Private Cloud endpoints or specific VPCs. Effectively, this

isolates network access to a given Amazon ECS resource from only the specific VPC within the AWS network. For more information, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\)](#) (p. 377).

## 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. AWS PrivateLink restricts all network traffic between your VPC and Amazon ECS to the Amazon network. You don't need an internet gateway, a NAT device, or a virtual private gateway.

For more information about AWS PrivateLink and VPC endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*.

### Considerations 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. 358).

- To allow your tasks to pull sensitive data from Secrets Manager, you must create the interface VPC endpoints for Secrets Manager. For more information, see [Using Secrets Manager with VPC Endpoints](#) in the *AWS Secrets Manager User Guide*.
- If your VPC doesn't have an internet gateway and your tasks use the `awslogs` log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#) in the *Amazon CloudWatch Logs User Guide*.
- VPC endpoints currently don't support cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to Amazon ECS.
- 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.

### 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.

## 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 \(p. 176\)](#).

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*.

## Create the Systems Manager Session Manager VPC endpoints when using the ECS Exec feature

If you use the ECS Exec feature, you need to create the interface VPC endpoints for Systems Manager Session Manager. For more information, see [Using Amazon ECS Exec for debugging \(p. 433\)](#).

For more information about Systems Manager Session Manager VPC endpoints, see [Use AWS PrivateLink to set up a VPC endpoint for Session Manager](#) in the *AWS Systems Manager User Guide*.

## Creating a VPC endpoint policy for Amazon ECS

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon ECS. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

### Example: VPC endpoint policy for Amazon ECS actions

The following is an example of an endpoint policy for Amazon ECS. When attached to an endpoint, this policy grants access to permission to create and list clusters. The `CreateCluster` and `ListClusters` actions do not accept any resources, so the resource definition is set to `*` for all resources.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

```
    ],  
    "Resource": [  
        "*"   
    ]  
  }  
]  
}
```

# Amazon ECS task metadata endpoint

## Important

If you are using Amazon ECS tasks hosted on Amazon EC2 instances, see [Amazon ECS task metadata endpoint](#) in the *Amazon Elastic Container Service Developer Guide*.

Amazon ECS on Fargate provides a method to retrieve various metadata, network metrics, and [Docker stats](#) about your containers and the tasks they are a part of. This is referred to as the *task metadata endpoint*. The following task metadata endpoint versions are available for Amazon ECS on Fargate tasks:

- Task metadata endpoint version 4 – Available for tasks that use platform version 1.4.0 or later.
- Task metadata endpoint version 3 – Available for tasks that use platform version 1.1.0 or later.

All containers belonging to tasks that are launched with the `awsvpc` network mode receive a local IPv4 address within a predefined link-local address range. When a container queries the metadata endpoint, the container agent can determine which task the container belongs to based on its unique IP address, and metadata and stats for that task are returned.

## Topics

- [Task metadata endpoint version 4](#) (p. 380)
- [Task metadata endpoint version 3](#) (p. 389)

## Task metadata endpoint version 4

### Important

If you are using Amazon ECS tasks hosted on Amazon EC2 instances, see [Amazon ECS task metadata endpoint](#) in the *Amazon Elastic Container Service Developer Guide*.

Beginning with Fargate platform version 1.4.0, an environment variable named `ECS_CONTAINER_METADATA_URI_V4` is injected into each container in a task. When you query the task metadata endpoint version 4, various task metadata and [Docker stats](#) are available to tasks.

The task metadata endpoint version 4 functions like the version 3 endpoint but provides additional network metadata for your containers and tasks. Additional network metrics are available when querying the `/stats` endpoints as well.

### Note

To avoid the need to create new task metadata endpoint versions in the future, additional metadata may be added to the version 4 output. We will not remove any existing metadata or change the metadata field names.

## Enabling the task metadata endpoint

The task metadata endpoint is enabled by default for all Amazon ECS tasks run on AWS Fargate that use platform version 1.4.0 or later.

## Task metadata endpoint version 4 paths

The following task metadata endpoints are available to containers:

`${ECS_CONTAINER_METADATA_URI_V4}`

This path returns metadata for the container.

`${ECS_CONTAINER_METADATA_URI_V4}/task`

This path returns metadata for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see [Task metadata JSON response \(p. 381\)](#).

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

This path returns Docker stats for the Docker container. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

**Note**

Amazon ECS tasks on AWS Fargate require that the container run for ~1 second prior to returning the container stats.

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

This path returns Docker stats for all of the containers associated with the task. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

**Note**

Amazon ECS tasks on AWS Fargate require that the container run for ~1 second prior to returning the container stats.

## Task metadata JSON response

The following metadata is returned in the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON response.

**Cluster**

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

**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 (Linux) or 1.0.0 or later (Windows).

**Containers**

A list of container metadata for each container associated with the task.

**DockerId**

The Docker ID for the container.

**Name**

The name of the container as specified in the task definition.

**DockerName**

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

**Image**

The image for the container.

**ImageID**

The SHA-256 digest 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.

#### ClockDrift

The information about the difference between the reference time and the system time. This applies to the Linux operating system.

#### ReferenceTime

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example `2021-09-07T16:57:44Z`.

#### ClockErrorBound

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

#### ClockSynchronizationStatus

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.

The valid values are `SYNCHRONIZED` and `NOT_SYNCHRONIZED`.

#### ExecutionStoppedAt

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

## Examples

The following examples show sample outputs from the task metadata endpoints for Amazon ECS tasks run on AWS Fargate.

You can use `curl` following by the task meta data endpoint to query the endpoint for example `curl ${ECS_CONTAINER_METADATA_URI_V4}/task`.

### Example container metadata response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}` endpoint you are returned only metadata about the container itself. The following is an example output.

```
{
  "DockerId": "cd189a933e5849daa93386466019ab50-2495160603",
  "Name": "curl",
  "DockerName": "curl",
  "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
  "ImageID": "sha256:25f3695bedfb454a50f12d127839a68ad3caf91e451c1da073db34c542c4d2cb",
  "Labels": {
```



```

        "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/
cd189a933e5849daa93386466019ab50",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "2"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-08T20:09:11.44527186Z",
    "StartedAt": "2020-10-08T20:09:11.44527186Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "192.0.2.3"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0a:de:f6:10:51:e5",
            "IPv4SubnetCIDRBlock": "192.0.2.0/24",
            "DomainNameServers": [
                "192.0.2.2"
            ],
            "DomainNameSearchList": [
                "us-west-2.compute.internal"
            ],
            "PrivateDNSName": "ip-10-0-0-222.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "192.0.2.0/24"
        }
    ],
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/05966557-
f16c-49cb-9352-24b3a0dcd0e1",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/containerlogs",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/cd189a933e5849daa93386466019ab50"
    },
    "LogDriver": "awslogs"
}

```

## 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-2:111122223333:cluster/default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/default/
e9028f8d5d8e4f258373e7b93ce9a3c3",
    "Family": "curltest",
    "Revision": "3",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 0.25,
        "Memory": 512
    },
    "PullStartedAt": "2020-10-08T20:47:16.053330955Z",
}

```

```

"PullStoppedAt": "2020-10-08T20:47:19.592684631Z",
"AvailabilityZone": "us-west-2a",
"Containers": [
  {
    "DockerId": "e9028f8d5d8e4f258373e7b93ce9a3c3-2495160603",
    "Name": "curl",
    "DockerName": "curl",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID":
      "sha256:25f3695bedfb454a50f12d127839a68ad3caf91e451c1da073db34c542c4d2cb",
    "Labels": {
      "com.amazonaws.ecs.cluster": "arn:aws:ecs:us-west-2:111122223333:cluster/
default",
      "com.amazonaws.ecs.container-name": "curl",
      "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/e9028f8d5d8e4f258373e7b93ce9a3c3",
      "com.amazonaws.ecs.task-definition-family": "curltest",
      "com.amazonaws.ecs.task-definition-version": "3"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
      "CPU": 10,
      "Memory": 128
    },
    "CreatedAt": "2020-10-08T20:47:20.567813946Z",
    "StartedAt": "2020-10-08T20:47:20.567813946Z",
    "Type": "NORMAL",
    "Networks": [
      {
        "NetworkMode": "awsvpc",
        "IPv4Addresses": [
          "192.0.2.3"
        ],
        "IPv6Addresses": [
          "2001:db8:10b:1a00:32bf:a372:d80f:e958"
        ],
        "AttachmentIndex": 0,
        "MACAddress": "02:b7:20:19:72:39",
        "IPv4SubnetCIDRBlock": "192.0.2.0/24",
        "IPv6SubnetCIDRBlock": "2600:1f13:10b:1a00::/64",
        "DomainNameServers": [
          "192.0.2.2"
        ],
        "DomainNameSearchList": [
          "us-west-2.compute.internal"
        ],
        "PrivateDNSName": "ip-172-31-30-173.us-west-2.compute.internal",
        "SubnetGatewayIpv4Address": "192.0.2.0/24"
      }
    ],
    "ClockDrift": {
      "ClockErrorBound": 0.5458234999999999,
      "ReferenceTimestamp": "2021-09-07T16:57:44Z",
      "ClockSynchronizationStatus": "SYNCHRONIZED"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/1bdcca8b-
f905-4ee6-885c-4064cb70f6e6",
    "LogOptions": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/containerlogs",
      "awslogs-region": "us-west-2",
      "awslogs-stream": "ecs/curl/e9028f8d5d8e4f258373e7b93ce9a3c3"
    },
    "LogDriver": "awslogs"
  }
]

```

```
    ],
    "LaunchType": "FARGATE"
}
```

## Example task stats response

When querying the `${ECS_CONTAINER_METADATA_URI_V4}/task/stats` endpoint you are returned network metrics about the task the container is part of. The following is an example output.

```
{
  "3d1f891cded94dc795608466cce8ddcf-464223573": {
    "read": "2020-10-08T21:24:44.938937019Z",
    "preread": "2020-10-08T21:24:34.938633969Z",
    "pids_stats": {},
    "blkio_stats": {
      "io_service_bytes_recursive": [
        {
          "major": 202,
          "minor": 26368,
          "op": "Read",
          "value": 638976
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Write",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Sync",
          "value": 638976
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Async",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Total",
          "value": 638976
        }
      ],
      "io_serviced_recursive": [
        {
          "major": 202,
          "minor": 26368,
          "op": "Read",
          "value": 12
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Write",
          "value": 0
        },
        {
          "major": 202,
          "minor": 26368,
          "op": "Sync",

```

```

        "value": 12
    },
    {
        "major": 202,
        "minor": 26368,
        "op": "Async",
        "value": 0
    },
    {
        "major": 202,
        "minor": 26368,
        "op": "Total",
        "value": 12
    }
],
"io_queue_recursive": [],
"io_service_time_recursive": [],
"io_wait_time_recursive": [],
"io_merged_recursive": [],
"io_time_recursive": [],
"sectors_recursive": []
},
"num_procs": 0,
"storage_stats": {},
"cpu_stats": {
    "cpu_usage": {
        "total_usage": 1137691504,
        "percpu_usage": [
            696479228,
            441212276,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0
        ],
        "usage_in_kernelmode": 80000000,
        "usage_in_usermode": 810000000
    },
    "system_cpu_usage": 9393210000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"precpu_stats": {
    "cpu_usage": {
        "total_usage": 1136624601,
        "percpu_usage": [
            695639662,
            440984939,
            0,
            0,
            0,
            0,
            0,
            0
        ]
    }
}

```

```

        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0
    ],
    "usage_in_kernelmode": 80000000,
    "usage_in_usermode": 810000000
},
"system_cpu_usage": 9373330000000,
"online_cpus": 2,
"throttling_data": {
    "periods": 0,
    "throttled_periods": 0,
    "throttled_time": 0
}
},
"memory_stats": {
    "usage": 6504448,
    "max_usage": 8458240,
    "stats": {
        "active_anon": 1675264,
        "active_file": 557056,
        "cache": 651264,
        "dirty": 0,
        "hierarchical_memory_limit": 536870912,
        "hierarchical_memsw_limit": 9223372036854772000,
        "inactive_anon": 0,
        "inactive_file": 3088384,
        "mapped_file": 430080,
        "pgfault": 11034,
        "pgmajfault": 5,
        "pgpgin": 8436,
        "pgpgout": 7137,
        "rss": 4669440,
        "rss_huge": 0,
        "total_active_anon": 1675264,
        "total_active_file": 557056,
        "total_cache": 651264,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 3088384,
        "total_mapped_file": 430080,
        "total_pgfault": 11034,
        "total_pgmajfault": 5,
        "total_pgpgin": 8436,
        "total_pgpgout": 7137,
        "total_rss": 4669440,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
    },
    "limit": 9223372036854772000
},
"name": "curltest",
"id": "3d1f891cded94dc795608466cce8ddcf-464223573",
"networks": {
    "eth1": {
        "rx_bytes": 2398415937,
        "rx_packets": 1898631,
        "rx_errors": 0,

```

```
    "rx_dropped": 0,  
    "tx_bytes": 1259037719,  
    "tx_packets": 428002,  
    "tx_errors": 0,  
    "tx_dropped": 0  
  }  
},  
"network_rate_stats": {  
  "rx_bytes_per_sec": 43.298687872232854,  
  "tx_bytes_per_sec": 215.39347269466413  
}  
}  
}
```

## Task metadata endpoint version 3

### Important

If you are using Amazon ECS tasks hosted on Amazon EC2 instances, see [Amazon ECS task metadata endpoint](#) in the *Amazon Elastic Container Service Developer Guide*.

Beginning with Fargate platform version 1.1.0, an environment variable named `ECS_CONTAINER_METADATA_URI` is injected into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and [Docker stats](#) are available to tasks.

## Enabling Task Metadata

The task metadata endpoint feature is enabled by default for Amazon ECS tasks hosted on Fargate that use platform version 1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 58\)](#).

## Task Metadata Endpoint Paths

The following API endpoints are available to containers:

`${ECS_CONTAINER_METADATA_URI}`

This path returns metadata JSON for the container.

`${ECS_CONTAINER_METADATA_URI}/task`

This path returns metadata JSON for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see [Task Metadata JSON Response \(p. 389\)](#).

`${ECS_CONTAINER_METADATA_URI}/stats`

This path returns Docker stats JSON for the specific Docker container. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

`${ECS_CONTAINER_METADATA_URI}/task/stats`

This path returns Docker stats JSON for all of the containers associated with the task. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

## Task Metadata JSON Response

The following information is returned from the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI}/task`) JSON response.

**Cluster**

The Amazon Resource Name (ARN) or short name 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 (Linux) or 1.0.0 or later (Windows).

**Containers**

A list of container metadata for each container associated with the task.

**DockerId**

The Docker ID for the container.

**Name**

The name of the container as specified in the task definition.

**DockerName**

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

**Image**

The image for the container.

**ImageID**

The SHA-256 digest 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.

**ClockDrift**

The information about the difference between the reference time and the system time. This applies to the Linux operating system.

**ReferenceTime**

The basis of clock accuracy. Amazon ECS uses the Coordinated Universal Time (UTC) global standard through NTP, for example 2021-09-07T16:57:44Z.

**ClockErrorBound**

The measure of clock error, defined as the offset to UTC. This error is the difference in milliseconds between the reference time and the system time.

**ClockSynchronizationStatus**

Indicates whether the most recent synchronization attempt between the system time and the reference time was successful.



The valid values are `SYNCHRONIZED` and `NOT_SYNCHRONIZED`.

#### ExecutionStoppedAt

The time stamp for when the tasks `DesiredStatus` moved to `STOPPED`. This occurs when an essential container moves to `STOPPED`.

## Example Task Metadata Response

The following JSON response is for a single-container task.

```
{
  "Cluster": "default",
  "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
  "Family": "nginx",
  "Revision": "5",
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Containers": [
    {
      "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
      "Name": "~internal-ecs-pause",
      "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
      "Image": "amazon/amazon-ecs-pause:0.1.0",
      "ImageID": "",
      "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "~internal-ecs-pause",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
      },
      "DesiredStatus": "RESOURCES_PROVISIONED",
      "KnownStatus": "RESOURCES_PROVISIONED",
      "Limits": {
        "CPU": 0,
        "Memory": 0
      },
      "CreatedAt": "2018-02-01T20:55:08.366329616Z",
      "StartedAt": "2018-02-01T20:55:09.058354915Z",
      "Type": "CNI_PAUSE",
      "Networks": [
        {
          "NetworkMode": "awsvpc",
          "IPv4Addresses": [
            "10.0.2.106"
          ]
        }
      ]
    }
  ],
  {
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
    "Name": "nginx-curl",
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
    "Image": "nrdlngr/nginx-curl",
    "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dcd458dc0de47bc165",
    "Labels": {
      "com.amazonaws.ecs.cluster": "default",
      "com.amazonaws.ecs.container-name": "nginx-curl",
      "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",

```

```
    "com.amazonaws.ecs.task-definition-family": "nginx",
    "com.amazonaws.ecs.task-definition-version": "5"
  },
  "DesiredStatus": "RUNNING",
  "KnownStatus": "RUNNING",
  "Limits": {
    "CPU": 512,
    "Memory": 512
  },
  "CreatedAt": "2018-02-01T20:55:10.554941919Z",
  "StartedAt": "2018-02-01T20:55:11.064236631Z",
  "Type": "NORMAL",
  "Networks": [
    {
      "NetworkMode": "awsvpc",
      "IPv4Addresses": [
        "10.0.2.106"
      ]
    }
  ]
},
"PullStartedAt": "2018-02-01T20:55:09.372495529Z",
"PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
"AvailabilityZone": "us-east-2b"
}
```

# AWS services integrated with Amazon ECS

Amazon ECS works with other AWS services to provide additional solutions for your business challenges. This topic identifies services that either use Amazon ECS to add functionality, or services that Amazon ECS uses to perform tasks.

## Contents

- [Using Amazon ECR with Amazon ECS \(p. 394\)](#)
- [Creating Amazon ECS resources with AWS CloudFormation \(p. 395\)](#)
- [Use App Mesh with Amazon ECS \(p. 395\)](#)

## Using Amazon ECR with Amazon ECS

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`.

## Creating Amazon ECS resources with AWS CloudFormation

Amazon ECS is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, for example an Amazon ECS cluster, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon ECS resources consistently and repeatedly. Just describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

### Amazon ECS and AWS CloudFormation templates

To provision and configure resources for Amazon ECS and related services, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the *AWS CloudFormation User Guide*.

Amazon ECS supports creating clusters, task definitions, services, and task sets in AWS CloudFormation. For more information, including examples of JSON and YAML templates for your Amazon ECS resources, see [Amazon ECS resource type reference](#) in the *AWS CloudFormation User Guide*.

### Learn more about AWS CloudFormation

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

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

## Use App Mesh with Amazon ECS

App Mesh is a service mesh that makes it easy to monitor and control services. App Mesh standardizes how your services communicate, giving you end-to-end visibility and helping to ensure high availability for your applications. App Mesh gives you consistent visibility and network traffic controls for every service in an application. You can get started using App Mesh with Amazon ECS by completing the [Getting started with AWS App Mesh and Amazon ECS](#) tutorial in the AWS App Mesh User Guide. The tutorial recommends that you have existing services deployed to Amazon ECS that you want to use App Mesh with.

**Note**

This feature is not available for Window containers on Fargate.

# Tutorials for Amazon ECS

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

## Topics

- [Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI \(p. 397\)](#)
- [Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI \(p. 403\)](#)
- [Tutorial: Specifying sensitive data using Secrets Manager secrets \(p. 408\)](#)
- [Tutorial: Creating a service using Service Discovery \(p. 413\)](#)
- [Tutorial: Creating a service using a blue/green deployment \(p. 420\)](#)
- [Tutorial: Listening for Amazon ECS CloudWatch Events \(p. 429\)](#)
- [Tutorial: Sending Amazon Simple Notification Service alerts for task stopped events \(p. 430\)](#)

## Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI

The following steps help you set up a cluster, register a task definition, run a Linux task, and perform other common scenarios in Amazon ECS with the AWS CLI. 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 \(p. 397\)](#)
- [Step 1: Create a Cluster \(p. 398\)](#)
- [Step 2: Register a Linux Task Definition \(p. 398\)](#)
- [Step 3: List Task Definitions \(p. 399\)](#)
- [Step 4: Create a Service \(p. 399\)](#)
- [Step 5: List Services \(p. 400\)](#)
- [Step 6: Describe the Running Service \(p. 400\)](#)
- [Step 7: Test \(p. 402\)](#)
- [Step 8: Clean Up \(p. 403\)](#)

## 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 [Set up to use Amazon ECS \(p. 4\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 333\)](#) IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Amazon ECR Public so your task must have internet access. To give your task a route to the internet, use one of the following options.

- Use a private subnet with a NAT gateway that has an elastic IP address.
- Use a public subnet and assign a public IP address to the task.

For more information, see [the section called “Create a virtual private cloud” \(p. 6\)](#).

- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

## Step 1: Create a Cluster

By default, your account receives a default cluster.

### Note

The benefit of using the default cluster that is provided for you is that you don't have to specify the `--cluster` *cluster\_name* option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster` *cluster\_name* for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{
  "cluster": {
    "status": "ACTIVE",
    "statistics": [],
    "clusterName": "fargate-cluster",
    "registeredContainerInstancesCount": 0,
    "pendingTasksCount": 0,
    "runningTasksCount": 0,
    "activeServicesCount": 0,
    "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
  }
}
```

## Step 2: Register a Linux Task Definition

Before you can run a task on your ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that creates a PHP web app using the httpd container image hosted on Docker Hub. For more information about the available task definition parameters, see [Amazon ECS task definitions \(p. 85\)](#).

```
{
  "family": "sample-fargate",
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "fargate-app",
      "image": "public.ecr.aws/docker/library/httpd:latest",
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp"
        }
      ]
    }
  ]
}
```

```
        }
      ],
      "essential": true,
      "entryPoint": [
        "sh",
        "-c"
      ],
      "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
<style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' > /usr/local/apache2/htdocs/index.html && httpd-foreground\""
      ]
    }
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "cpu": "256",
  "memory": "512"
}
```

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: Test

Describe the task in the service so that you can get the Elastic Network Interface (ENI) for the task.

Describe the task and locate the ENI ID.

```
aws ecs describe-tasks --cluster fargate-cluster --tasks fargate-tasks
```

The attachment information is listed in the output.

```
{
  "tasks": [
    {
      "attachments": [
        {
          "id": "d9e7735a-16aa-4128-bc7a-b2d5115029e9",
          "type": "ElasticNetworkInterface",
          "status": "ATTACHED",
          "details": [
            {
              "name": "subnetId",
              "value": "subnetabcd1234"
            },
            {
              "name": "networkInterfaceId",
              "value": "eni-0fa40520aeEXAMPLE"
            }
          ]
        }
      ]
    }
  ]
}
```

Describe the ENI to get the public IP address.

```
aws ec2 describe-network-interfaces --network-interface-id eni-0fa40520aeEXAMPLE
```

The public IP address is in the output.

```
{
  "NetworkInterfaces": [
    {
      "Association": {
```

```
        "IpOwnerId": "amazon",  
        "PublicDnsName": "ec2-34-229-42-222.compute-1.amazonaws.com",  
        "PublicIp": "198.51.100.2"  
    },  
    ...  
}
```

Enter the public IP address in your web browser and you should see a webpage that displays the **Amazon ECS** sample application.

## Step 8: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

# Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI

The following steps help you set up a cluster, register a task definition, run a Windows task, and perform other common scenarios in Amazon ECS with the AWS CLI. Ensure that you are using the latest version of the AWS CLI. For more information on how to upgrade to the latest version, see [Installing the AWS Command Line Interface](#).

### Topics

- [Prerequisites \(p. 403\)](#)
- [Step 1: Create a Cluster \(p. 404\)](#)
- [Step 2: Register a Windows Task Definition \(p. 404\)](#)
- [Step 3: List task definitions \(p. 405\)](#)
- [Step 4: Create a service \(p. 406\)](#)
- [Step 5: List services \(p. 406\)](#)
- [Step 6: Describe the Running Service \(p. 406\)](#)
- [Step 7: Clean Up \(p. 408\)](#)

## 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 [Set up to use Amazon ECS \(p. 4\)](#) have been completed.

- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions](#) (p. 333) IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Docker Hub so your task must have internet access. To give your task a route to the internet, use one of the following options.
  - Use a private subnet with a NAT gateway that has an elastic IP address.
  - Use a public subnet and assign a public IP address to the task.

For more information, see [the section called "Create a virtual private cloud"](#) (p. 6).

- Optional: AWS CloudShell is a tool that gives customers a command line without needing to create their own EC2 instance. For more information, see [What is AWS CloudShell](#) in the *AWS CloudShell User Guide*.

## Step 1: Create a Cluster

By default, your account receives a default cluster.

### Note

The benefit of using the default cluster that is provided for you is that you don't have to specify the `--cluster` *cluster\_name* option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster` *cluster\_name* for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{
  "cluster": {
    "status": "ACTIVE",
    "statistics": [],
    "clusterName": "fargate-cluster",
    "registeredContainerInstancesCount": 0,
    "pendingTasksCount": 0,
    "runningTasksCount": 0,
    "activeServicesCount": 0,
    "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"
  }
}
```

## Step 2: Register a Windows Task Definition

Before you can run a Windows task on your Amazon ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that creates a web app. For more information about the available task definition parameters, see [Amazon ECS task definitions](#) (p. 85).

```
{
  "containerDefinitions": [
    {
      "command": [
        "New-Item -Path C:\\inetpub\\wwwroot\\index.html -Type file -Value '<html>
<head> <title>Amazon ECS Sample App</title> <style>body {margin-top: 40px; background-
```

```
color: #333;} </style> </head><body> <div style=color:white;text-align:center> <h1>Amazon
ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a
container in Amazon ECS.</p>'; C:\\ServiceMonitor.exe w3svc"
    ],
    "entryPoint": [
        "powershell",
        "-Command"
    ],
    "essential": true,
    "cpu": 2048,
    "memory": 4096,
    "image": "mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019",
    "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
            "awslogs-group": "/ecs/fargate-windows-task-definition",
            "awslogs-region": "us-east-1",
            "awslogs-stream-prefix": "ecs"
        }
    },
    "name": "sample_windows_app",
    "portMappings": [
        {
            "hostPort": 80,
            "containerPort": 80,
            "protocol": "tcp"
        }
    ]
}

},
"memory": "4096",
"cpu": "2048",
"networkMode": "awsvpc",
"family": "windows-simple-iis-2019-core",
"executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",
"runtimePlatform": {
    "operatingSystemFamily": "WINDOWS_SERVER_2019_CORE"
},
"requiresCompatibilities": [
    "FARGATE"
]
}
```

The above example JSON can be passed to the AWS CLI in two ways: You can save the task definition JSON as a file and pass it with the `--cli-input-json file://path_to_file.json` option.

To use a JSON file for container definitions:

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

The **register-task-definition** command returns a description of the task definition after it completes its registration.

## Step 3: List task definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the `family` and `revision` values that you can use together when calling **run-task** or **start-task**.

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

Output:

```
{
  "taskDefinitionArns": [
    "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate-windows:1"
  ]
}
```

## Step 4: Create a service

After you have registered a task for your account, you can create a service for the registered task in your cluster. For this example, you create a service with one instance of the `sample-fargate:1` task definition running in your cluster. The task requires a route to the internet, so there are two ways you can achieve this. One way is to use a private subnet configured with a NAT gateway with an elastic IP address in a public subnet. Another way is to use a public subnet and assign a public IP address to your task. We provide both examples below.

Example using a private subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate-windows:1 --desired-count 1 --launch-type "FARGATE" --
network-configuration "awsVpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234]}"
```

Example using a public subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --
task-definition sample-fargate-windows:1 --desired-count 1 --launch-type "FARGATE" --
network-configuration "awsVpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-
abcd1234],assignPublicIp=ENABLED}"
```

The `create-service` command returns a description of the task definition after it completes its registration.

## Step 5: List services

List the services for your cluster. You should see the service that you created in the previous section. You can take the service name or the full ARN that is returned from this command and use it to describe the service later.

```
aws ecs list-services --cluster fargate-cluster
```

Output:

```
{
  "serviceArns": [
    "arn:aws:ecs:region:aws_account_id:service/fargate-service"
  ]
}
```

## Step 6: Describe the Running Service

Describe the service using the service name retrieved earlier to get more information about the task.

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

If successful, this will return a description of the service failures and services. For example, in services section, you will find information on deployments, such as the status of the tasks as running or pending. You may also find information on the task definition, the network configuration and time-stamped events. In the failures section, you will find information on failures, if any, associated with the call. For troubleshooting, see [Service Event Messages](#). For more information about the service description, see [Describe Services](#).

```
{
  "services": [
    {
      "status": "ACTIVE",
      "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate-windows:1",
      "pendingCount": 2,
      "launchType": "FARGATE",
      "loadBalancers": [],
      "roleArn": "arn:aws:iam::aws_account_id:role/aws-service-role/ecs.amazonaws.com/AWSServiceRoleForECS",
      "placementConstraints": [],
      "createdAt": 1510811361.128,
      "desiredCount": 2,
      "networkConfiguration": {
        "awsvpcConfiguration": {
          "subnets": [
            "subnet-abcd1234"
          ],
          "securityGroups": [
            "sg-abcd1234"
          ],
          "assignPublicIp": "DISABLED"
        }
      },
      "platformVersion": "LATEST",
      "serviceName": "fargate-service",
      "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster",
      "serviceArn": "arn:aws:ecs:region:aws_account_id:service/fargate-service",
      "deploymentConfiguration": {
        "maximumPercent": 200,
        "minimumHealthyPercent": 100
      },
      "deployments": [
        {
          "status": "PRIMARY",
          "networkConfiguration": {
            "awsvpcConfiguration": {
              "subnets": [
                "subnet-abcd1234"
              ],
              "securityGroups": [
                "sg-abcd1234"
              ],
              "assignPublicIp": "DISABLED"
            }
          },
          "pendingCount": 2,
          "launchType": "FARGATE",
          "createdAt": 1510811361.128,
          "desiredCount": 2,
          "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate-windows:1",
          "updatedAt": 1510811361.128,

```



```
        "platformVersion": "0.0.1",
        "id": "ecs-svc/9223370526043414679",
        "runningCount": 0
      },
      ],
      "events": [
        {
          "message": "(service fargate-service) has started 2 tasks: (task 53c0de40-ea3b-489f-a352-623bf1235f08) (task d0aec985-901b-488f-9fb4-61b991b332a3).",
          "id": "92b8443e-67fb-4886-880c-07e73383ea83",
          "createdAt": 1510811841.408
        },
        {
          "message": "(service fargate-service) has started 2 tasks: (task b4911bee-7203-4113-99d4-e89ba457c626) (task cc5853e3-6e2d-4678-8312-74f8a7d76474).",
          "id": "d85c6ec6-a693-43b3-904a-a997e1fc844d",
          "createdAt": 1510811601.938
        },
        {
          "message": "(service fargate-service) has started 2 tasks: (task cba86182-52bf-42d7-9df8-b744699e6cfc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
          "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
          "createdAt": 1510811364.691
        }
      ],
      "runningCount": 0,
      "placementStrategy": []
    },
    ],
    "failures": []
  }
}
```

## Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

## 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 \(p. 176\)](#).

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 [Set up to use Amazon ECS \(p. 4\)](#) 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 \(p. 354\)](#).

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 `ECSecretsTutorial`, 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**.

```
{
  "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
```

```
    "/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 **Select cluster compatibility**, choose **EC2 Linux + Networking**, then choose **Next step**.
6. On the **Configure cluster** page, for **Cluster name** enter `ecs-secrets-tutorial`.
7. For **EC2 instance type**, choose **t2.micro**.
8. 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.

9. In the **Networking** section, configure the VPC for your cluster. Select an existing VPC or you can choose **Create a new VPC** to use for the tutorial.
  - 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.
10. For **Container instance IAM role**, choose your existing container instance IAM role or select **Create new role** to have one created for you.

11. 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 \(p. 442\)](#) 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 is now integrated into the Create Service wizard in the Amazon ECS console. For more information, see [Creating an Amazon ECS service \(p. 226\)](#).

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 AWS Regions that support service discovery, see [Service Discovery \(p. 271\)](#).

For information about the Regions that support Fargate, see [the section called “AWS Fargate Regions” \(p. 285\)](#).

## Prerequisites

Before you start this tutorial, make sure that the following prerequisites are met:

- The latest version of the AWS CLI is installed and configured. For more information, see [Installing the AWS Command Line Interface](#).
- The steps described in [Set up to use Amazon ECS \(p. 4\)](#) are complete.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 333\)](#) IAM policy example.
- You have created at least one VPC and one security group. For more information, see [the section called “Create a virtual private cloud” \(p. 6\)](#).

## Step 1: Create the Service Discovery resources in AWS Cloud Map

Follow these steps to create your service discovery namespace and service discovery service:

1. Create a private Cloud Map service discovery namespace. This example creates a namespace that's called `tutorial`. Replace `vpc-abcd1234` with the ID of one of your existing VPCs.

```
aws servicediscovery create-private-dns-namespace \  
  --name tutorial \  
  --vpc vpc-abcd1234
```

The output of this command is as follows.

```
{  
  "OperationId": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e"  
}
```

2. Using the `OperationId` from the output of the previous step, verify that the private namespace was created successfully. Make note of the namespace ID because you use it in subsequent commands.

```
aws servicediscovery get-operation \  
  --operation-id h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e
```

The output is as follows.

```
{  
  "Operation": {  
    "Id": "h2qe3s6dxftvvt7riu6lfy2f6c3jlhf4-je6chs2e",  
    "Type": "CREATE_NAMESPACE",  
    "Status": "SUCCESS",  
    "CreateDate": 1519777852.502,  
    "UpdateDate": 1519777856.086,  
    "Targets": {  
      "NAMESPACE": "ns-uejictsjen2i4eeg"  
    }  
  }  
}
```

- Using the `NAMESPACE` ID from the output of the previous step, create a service discovery service. This example creates a service named `myapplication`. Make note of the service ID and ARN because you use them in subsequent commands.

```
aws servicediscovery create-service \  
  --name myapplication \  
  --dns-config "NamespaceId="ns-  
uejictsjen2i4eeg",DnsRecords=[{Type="A",TTL="300"}]" \  
  --health-check-custom-config FailureThreshold=1
```

The output is as follows.

```
{  
  "Service": {  
    "Id": "srv-utcrh6wavdkgggqtk",  
    "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-  
utcrh6wavdkgggqtk",  
    "Name": "myapplication",  
    "DnsConfig": {  
      "NamespaceId": "ns-uejictsjen2i4eeg",  
      "DnsRecords": [  
        {  
          "Type": "A",  
          "TTL": 300  
        }  
      ]  
    },  
    "HealthCheckCustomConfig": {  
      "FailureThreshold": 1  
    },  
    "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"  
  }  
}
```

## Step 2: Create the Amazon ECS resources

Follow these steps to create your Amazon ECS cluster, task definition, and service:

- Create an Amazon ECS cluster. This example creates a cluster that's named `tutorial`.

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

- Register a task definition that's compatible with Fargate and uses the `awsvpc` network mode. Follow these steps:
  - Create a file that's named `fargate-task.json` with the contents of the following task definition.

```
{  
  "family": "tutorial-task-def",  
  "networkMode": "awsvpc",  
  "containerDefinitions": [  
    {  
      "name": "sample-app",  
      "image": "httpd:2.4",  
      "portMappings": [  
        {  
          "containerPort": 80,  
          "hostPort": 80        }  
      ]  
    }  
  ]  
}
```



```
        "hostPort": 80,
        "protocol": "tcp"
      },
      ],
      "essential": true,
      "entryPoint": [
        "sh",
        "-c"
      ],
      "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample
App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</
h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in
Amazon ECS.</p> </div></body></html>' > /usr/local/apache2/htdocs/index.html &&
httpd-foreground\""
      ]
    }
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "cpu": "256",
  "memory": "512"
}
```

- b. Register the task definition using `fargate-task.json`.

```
aws ecs register-task-definition \
  --cli-input-json file://fargate-task.json
```

3. Create an ECS service by following these steps:

- a. Create a file that's named `ecs-service-discovery.json` with the contents of the ECS service that you're creating. This example uses the task definition that was created in the previous step. An `awsvpcConfiguration` is required because the example task definition uses the `awsvpc` network mode.

When you create the ECS service, specify the Fargate launch type, and the `LATEST` platform version that supports service discovery. When the service discovery service is created in AWS Cloud Map, `registryArn` is the ARN returned. The `securityGroups` and `subnets` must belong to the VPC that's used to create the Cloud Map namespace. You can obtain the security group and subnet IDs from the Amazon VPC Console.

```
{
  "cluster": "tutorial",
  "serviceName": "ecs-service-discovery",
  "taskDefinition": "tutorial-task-def",
  "serviceRegistries": [
    {
      "registryArn":
"arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkgggqtk"
    }
  ],
  "launchType": "FARGATE",
  "platformVersion": "LATEST",
  "networkConfiguration": {
    "awsvpcConfiguration": {
      "assignPublicIp": "ENABLED",
      "securityGroups": [ "sg-abcd1234" ],
      "subnets": [ "subnet-abcd1234" ]
    }
  }
},
```

```
}
  "desiredCount": 1
}
```

- b. Create your ECS service using `ecs-service-discovery.json`.

```
aws ecs create-service \
  --cli-input-json file://ecs-service-discovery.json
```

## Step 3: Verify Service Discovery in AWS Cloud Map

You can verify that everything is created properly by querying your service discovery information. After service discovery is configured, you can either use AWS Cloud Map API operations, or call `dig` from an instance within your VPC. Follow these steps:

1. Using the service discovery service ID, list the service discovery instances. Make note of the instance ID (marked in bold) for resource cleanup.

```
aws servicediscovery list-instances \
  --service-id srv-utcrh6wavdkggqtk
```

The output is as follows.

```
{
  "Instances": [
    {
      "Id": "16becc26-8558-4af1-9fbd-f81be062a266",
      "Attributes": {
        "AWS_INSTANCE_IPV4": "172.31.87.2",
        "AWS_INSTANCE_PORT": "80",
        "AVAILABILITY_ZONE": "us-east-1a",
        "REGION": "us-east-1",
        "ECS_SERVICE_NAME": "ecs-service-discovery",
        "ECS_CLUSTER_NAME": "tutorial",
        "ECS_TASK_DEFINITION_FAMILY": "tutorial-task-def"
      }
    }
  ]
}
```

2. Use the service discovery namespace, service, and additional parameters such as ECS cluster name to query details about the service discovery instances.

```
aws servicediscovery discover-instances \
  --namespace-name tutorial \
  --service-name myapplication \
  --query-parameters ECS_CLUSTER_NAME=tutorial
```

3. The DNS records that are created in the Route 53 hosted zone for the service discovery service can be queried with the following AWS CLI commands:
  - a. Using the namespace ID, get information about the namespace, which includes the Route 53 hosted zone ID.

```
aws servicediscovery \
  get-namespace --id ns-uejictsjen2i4eeg
```

The output is as follows.

```
{
  "Namespace": {
    "Id": "ns-uejictsjen2i4eeg",
    "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-uejictsjen2i4eeg",
    "Name": "tutorial",
    "Type": "DNS_PRIVATE",
    "Properties": {
      "DnsProperties": {
        "HostedZoneId": "Z35JQ4ZFDRYPLV"
      }
    },
    "CreateDate": 1519777852.502,
    "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"
  }
}
```

- b. Using the Route 53 hosted zone ID from the previous step (see the text in bold), get the resource record set for the hosted zone.

```
aws route53 list-resource-record-sets \
  --hosted-zone-id Z35JQ4ZFDRYPLV
```

4. You can also query the DNS from an instance within your VPC using dig.

```
dig +short myapplication.tutorial
```

## Step 4: Clean up

When you're finished with this tutorial, clean up the associated resources to avoid incurring charges for unused resources. Follow these steps:

1. Deregister the service discovery service instances using the service ID and instance ID that you noted previously.

```
aws servicediscovery deregister-instance \
  --service-id srv-utcrh6wavdkggqtk \
  --instance-id 16becc26-8558-4af1-9fbd-f81be062a266
```

The output is as follows.

```
{
  "OperationId": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv"
}
```

2. Using the OperationId from the output of the previous step, verify that the service discovery service instances were deregistered successfully.

```
aws servicediscovery get-operation \
  --operation-id xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv
```

```
{
  "Operation": {
    "Id": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv",
    "Type": "DEREGISTER_INSTANCE",
    "Status": "SUCCESS",
  }
}
```

```
    "CreateDate": 1525984073.707,
    "UpdateDate": 1525984076.426,
    "Targets": {
      "INSTANCE": "16becc26-8558-4af1-9fbd-f81be062a266",
      "ROUTE_53_CHANGE_ID": "C5NSRG1J4I1FH",
      "SERVICE": "srv-utcrh6wavdkgggtk"
    }
  }
}
```

3. Delete the service discovery service using the service ID.

```
aws servicediscovery delete-service \
  --id srv-utcrh6wavdkgggtk
```

4. Delete the service discovery namespace using the namespace ID.

```
aws servicediscovery delete-namespace \
  --id ns-uejictsjen2i4eeg
```

The output is as follows.

```
{
  "OperationId": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj"
}
```

5. Using the OperationId from the output of the previous step, verify that the service discovery namespace was deleted successfully.

```
aws servicediscovery get-operation \
  --operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj
```

The output is as follows.

```
{
  "Operation": {
    "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
    "Type": "DELETE_NAMESPACE",
    "Status": "SUCCESS",
    "CreateDate": 1525984602.211,
    "UpdateDate": 1525984602.558,
    "Targets": {
      "NAMESPACE": "ns-rymlehshst7hhukh",
      "ROUTE_53_CHANGE_ID": "CJP2A2M86XW3O"
    }
  }
}
```

6. Update the desired count for the Amazon ECS service to 0. You must do this to delete the service in the next step.

```
aws ecs update-service \
  --cluster tutorial \
  --service ecs-service-discovery \
  --desired-count 0
```

7. Delete the Amazon ECS service.

```
aws ecs delete-service \
```

```
--cluster tutorial \  
--service ecs-service-discovery
```

8. Delete the Amazon ECS cluster.

```
aws ecs delete-cluster \  
--cluster tutorial
```

## Tutorial: Creating a service using a blue/green deployment

Amazon ECS has integrated blue/green deployments into the Create Service wizard on the Amazon ECS console. For more information, see [Creating an Amazon ECS service \(p. 226\)](#).

The following tutorial shows how to create an Amazon ECS service containing a Fargate task that uses the blue/green deployment type with the AWS CLI.

### Note

Support for performing a blue/green deployment has been added for AWS CloudFormation. For more information, see [Perform Amazon ECS blue/green deployments through CodeDeploy using AWS CloudFormation](#) in the *AWS CloudFormation User Guide*.

## Prerequisites

This tutorial assumes that you have completed the following prerequisites:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading the AWS CLI, see [Installing the AWS Command Line Interface](#).
- The steps in [Set up to use Amazon ECS \(p. 4\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 333\)](#) IAM policy example.
- You have a VPC and security group created to use. For more information, see [the section called “Create a virtual private cloud” \(p. 6\)](#).
- The Amazon ECS CodeDeploy IAM role is created. For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 365\)](#).

## 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. 354).

```
{
  "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/ec2TaskExecutionRole"
}
```

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 a 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 a 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. 365).

```
{  
  "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 a 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 using the classic console \(p. 77\)](#) to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Lambda function correctly.

## Step 1: Create the Lambda function

In this procedure, you create a simple Lambda function to serve as a target for Amazon ECS event stream messages.

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose **Create function**.
3. On the **Author from scratch** screen, do the following:
  - a. For **Name**, enter a value.
  - b. For **Runtime**, choose your version of Python, for example, **Python 3.9**.
  - c. For **Role**, choose **Create a new role with basic Lambda permissions**.
4. Choose **Create function**.
5. In the **Function code** section, edit the sample code to match the following example:

```
import json

def lambda_handler(event, context):
    if event["source"] != "aws.ecs":
        raise ValueError("Function only supports input from events with a source type of: aws.ecs")

    print('Here is the event:')
    print(json.dumps(event))
```

This is a simple Python 3.9 function that prints the event sent by Amazon ECS. If everything is configured correctly, at the end of this tutorial, you see that the event details appear in the CloudWatch Logs log stream associated with this Lambda function.

6. Choose **Save**.

## Step 2: Register an event rule

Next, you create a CloudWatch Events event rule that captures task events coming from your Amazon ECS clusters. This rule captures all events coming from all clusters within the account where it is defined. The task messages themselves contain information about the event source, including the cluster on which it resides, that you can use to filter and sort events programmatically.

### Note

When you use the AWS Management Console to create an event rule, the console automatically adds the IAM permissions necessary to grant CloudWatch Events permission to call your Lambda

function. If you are creating an event rule using the AWS CLI, you need to grant this permission explicitly. For more information, see [Events 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 an Amazon EventBridge event rule that only captures task events where the task has stopped running because one of its essential containers has terminated. The event sends only task events with a specific `stoppedReason` property to the designated Amazon SNS topic.

### Prerequisite: Set up a test cluster

If you do not have a running cluster to capture events from, follow the steps in [Creating a cluster using the classic console \(p. 77\)](#) to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Amazon SNS topic and EventBridge rule correctly.

## Step 1: Create and subscribe to an Amazon SNS topic

For this tutorial, you configure an Amazon SNS topic to serve as an event target for your new event rule.

For information about how to create and subscribe to an Amazon SNS topic , see [Getting started with Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide* and use the following table to determine what options to select.

Option	Value	
Type	Standard	
Name	TaskStoppedAlert	
Protocol	Email	
Endpoint	An email address to which you currently have access	

## Step 2: Register an event rule

Next, you register an event rule that captures only task-stopped events for tasks with stopped containers.

For information about how to create and subscribe to an Amazon SNS topic , see [Create a rule in Amazon EventBridge](#) in the *Amazon EventBridge User Guide* and use the following table to determine what options to select.

Option	Value	
Rule type	Rule with an event pattern	
Event source	AWS events or EventBridge partner events	
Event pattern	Custom pattern (JSON editor)	
Event pattern	<pre>{   "source": [     "aws.ecs"   ],   "detail-type": [     "ECS Task State Change"   ],   "detail": {     "lastStatus": [       "STOPPED"     ],     "stoppedReason": [       "Essential container in task exited"     ]   } }</pre>	
Target type	AWS service	
Target	SNS topic	
Topic	TaskStoppedAlert (The topic you created in Step 1)	



## Step 3: Test your rule

Verify that the rule is working by running a task that exits shortly after it starts. If your event rule is configured correctly, you receive an email message within a few minutes with the event text. If you have an existing task definition that can satisfy the rule requirements, run a task using it. If you do not, the following steps will walk you through registering a Fargate task definition and running it that will.

### 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.

# 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

- [Using Amazon ECS Exec for debugging \(p. 433\)](#)
- [Checking stopped tasks for errors \(p. 442\)](#)
- [Stopped tasks error codes \(p. 444\)](#)
- [CannotPullContainer task errors \(p. 447\)](#)
- [Service event messages \(p. 450\)](#)
- [Invalid CPU or memory value specified \(p. 453\)](#)
- [Troubleshooting service load balancers \(p. 454\)](#)
- [Troubleshooting service auto scaling \(p. 455\)](#)
- [AWS Fargate throttling quotas \(p. 455\)](#)
- [API failure reasons \(p. 456\)](#)

## Using Amazon ECS Exec for debugging

With Amazon ECS Exec, you can directly interact with containers without needing to first interact with the host container operating system, open inbound ports, or manage SSH keys. You can use ECS Exec to run commands in or get a shell to a container running on an Amazon EC2 instance or on AWS Fargate. This makes it easier to collect diagnostic information and quickly troubleshoot errors. For example, in a development context, you can use ECS Exec to easily interact with various process in your containers and troubleshoot your applications. And, in production scenarios, you can use it to gain break-glass access to your containers to debug issues.

You can run commands in a running Linux or Windows container using ECS Exec from the Amazon ECS API, AWS Command Line Interface (AWS CLI), AWS SDKs, or the AWS Copilot CLI. For details on using ECS Exec, as well as a video walkthrough, using the AWS Copilot CLI, see the [Copilot Github documentation](#).

You can also use ECS Exec to maintain stricter access control policies and audit container access. By selectively turning on this feature, you can control who can run commands and on which tasks they can run those commands. With a log of each command and their output, you can use ECS Exec to audit which tasks were run and you can use CloudTrail to audit who accessed a container.

## Architecture

ECS Exec makes use of AWS Systems Manager (SSM) Session Manager to establish a connection with the running container and uses AWS Identity and Access Management (IAM) policies to control access to running commands in a running container. This is made possible by bind-mounting the necessary SSM agent binaries into the container. The Amazon ECS or AWS Fargate agent is responsible for starting the SSM core agent inside the container alongside your application code. For more information, see [Systems Manager Session Manager](#).

You can audit which user accessed the container using AWS CloudTrail and log each command (and their output) to Amazon S3 or Amazon CloudWatch Logs. To encrypt data between the local client and

container with your own encryption key, you must provide the AWS Key Management Service (AWS KMS) key.

## Considerations for using ECS Exec

For this topic, you should be familiar with the following aspects involved with using ECS Exec:

- ECS Exec is supported for AWS Fargate, external instances (ECS Anywhere), Linux containers hosted on Amazon EC2 and the following Windows Amazon ECS-optimized AMIs (with the container agent version 1.56 or later):
  - Amazon ECS-optimized Windows Server 2022 Full AMI
  - Amazon ECS-optimized Windows Server 2022 Core AMI
  - Amazon ECS-optimized Windows Server 2019 Full AMI
  - Amazon ECS-optimized Windows Server 2019 Core AMI
  - Amazon ECS-optimized Windows Server 20H2 Core AMI
- ECS Exec is not currently supported using the AWS Management Console.
- ECS Exec is not currently supported for tasks launched using an Auto Scaling group capacity provider.
- If you are using interface Amazon VPC endpoints with Amazon ECS, you must create the interface Amazon VPC endpoints for Systems Manager Session Manager. For more information, see [Create the Systems Manager Session Manager VPC endpoints when using the ECS Exec feature \(p. 378\)](#).
- You can't enable ECS Exec for existing tasks. It can only be enabled for new tasks.
- When a user runs commands on a container using ECS Exec, these commands are run as the `root` user. The SSM agent and its child processes run as root even when you specify a user ID for the container.
- The ECS Exec session has a default idle timeout time of 20 minutes. For more information, see [Specify an idle session timeout value](#) in the *AWS Systems Manager User Guide*.
- The SSM agent requires that the container file system is able to be written to in order to create the required directories and files. Therefore, making the root file system read-only using the `readonlyRootFilesystem` task definition parameter, or any other method, isn't supported.
- Users can run all of the commands that are available within the container context. The following actions might result in orphaned and zombie processes: terminating the main process of the container, terminating the command agent, and deleting dependencies. To cleanup zombie processes, we recommend adding the `initProcessEnabled` flag to your task definition.
- While starting SSM sessions outside of the `execute-command` action is possible, this results in the sessions not being logged and being counted against the session limit. We recommend limiting this access by denying the `ssm:start-session` action using an IAM policy. For more information, see [Limiting access to the Start Session action \(p. 441\)](#).
- ECS Exec will use some CPU and memory. You'll want to accommodate for that when specifying the CPU and memory resource allocations in your task definition.
- You must be using AWS CLI version 1.22.3 or later or AWS CLI version 2.3.6 or later. For information about how to update the AWS CLI, see [Installing or updating the latest version of the AWS CLI](#) in the *AWS Command Line Interface User Guide Version 2*.
- You cannot use ECS Exec when you use `run-task` to launch a task on a cluster that uses managed scaling with asynchronous placement (launch a task with no instance).
- You cannot run ECS Exec against Microsoft Nano Server containers. For more information about Nano Server containers, see [Nano Server](#) on the Docker web site.

## Prerequisites for using ECS Exec

Before you start using ECS Exec, make sure you that you have completed these actions:

- Install and configure the AWS CLI. For more information, see [AWS CLI](#).
- Install Session Manager plugin for the AWS CLI. For more information, see [Install the Session Manager plugin for the AWS CLI](#).
- ECS Exec has version requirements depending on whether your tasks are hosted on Amazon EC2 or AWS Fargate:
  - If you're using Amazon EC2, you must use an Amazon ECS optimized AMI that was released after January 20th, 2021, with an agent version of 1.50.2 or greater. For more information, see [Amazon ECS optimized AMIs](#).
  - If you're using AWS Fargate, you must use platform version 1.4.0 or higher (Linux) or 1.0.0 (Windows). For more information, see [AWS Fargate platform versions](#).

## Enabling and using ECS Exec

### IAM permissions required for ECS Exec

The ECS Exec feature requires a task IAM role to grant containers the permissions needed for communication between the managed SSM agent (`execute-command` agent) and the SSM service. For more information, see [Amazon ECS task IAM role](#). You should add the following permissions to a task IAM role and include the task IAM role in your task definition. For more information, see [Adding and Removing IAM Policies](#).

Use the following policy for your task IAM role to add the required SSM permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateControlChannel",
        "ssmmessages:CreateDataChannel",
        "ssmmessages:OpenControlChannel",
        "ssmmessages:OpenDataChannel"
      ],
      "Resource": "*"
    }
  ]
}
```

### Optional task definition changes

If you set the task definition parameter `initProcessEnabled` to `true`, this starts the init process inside the container, which removes any zombie SSM agent child processes found. The following provides an example.

```
{
  "taskRoleArn": "ecsTaskRole",
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
    "EC2",
    "FARGATE"
  ],
  "executionRoleArn": "ecsTaskExecutionRole",
  "memory": ".5 gb",
  "cpu": ".25 vcpu",
  "containerDefinitions": [
```

```
{
  "name": "amazon-linux",
  "image": "amazonlinux:latest",
  "essential": true,
  "command": ["sleep", "3600"],
  "linuxParameters": {
    "initProcessEnabled": true
  }
},
"family": "ecs-exec-task"
}
```

## Enabling ECS Exec for your tasks and services

You can enable the ECS Exec feature for your services and standalone tasks by specifying the `--enable-execute-command` flag when using one of the following AWS CLI commands: [create-service](#), [update-service](#), [start-task](#), or [run-task](#).

For example, if you run the following command, the ECS Exec feature is enabled for a newly created service. For more information about creating services, see [create-service](#).

```
aws ecs create-service \
  --cluster cluster-name \
  --task-definition task-definition-name \
  --enable-execute-command \
  --service-name service-name \
  --desired-count 1
```

After you have enabled ECS Exec for a task, you can run the following command to confirm the task is ready to be used. If the `lastStatus` property of the `ExecuteCommandAgent` is listed as `RUNNING` and the `enableExecuteCommand` property is set to `true`, then your task is ready.

```
aws ecs describe-tasks \
  --cluster cluster-name \
  --tasks task-id
```

The following output snippet is an example of what you might see.

```
{
  "tasks": [
    {
      ...
      "containers": [
        {
          ...
          "managedAgents": [
            {
              "lastStartedAt": "2021-03-01T14:49:44.574000-06:00",
              "name": "ExecuteCommandAgent",
              "lastStatus": "RUNNING"
            }
          ]
        }
      ],
      ...
      "enableExecuteCommand": true,
      ...
    }
  ]
}
```

```
}
```

## Running commands using ECS Exec

After you have confirmed the `ExecuteCommandAgent` is running, you can open an interactive shell on your container using the following command. If your task contains multiple containers, you must specify the container name using the `--container` flag. Amazon ECS only supports initiating interactive sessions, so you must use the `--interactive` flag.

The following command will run an interactive `/bin/sh` command against a container named `container-name` for a task with an id of `task-id`.

```
aws ecs execute-command --cluster cluster-name \
  --task task-id \
  --container container-name \
  --interactive \
  --command "/bin/sh"
```

## Logging and Auditing using ECS Exec

### Enabling logging and auditing in your tasks and services

Amazon ECS provides a default configuration for logging commands run using ECS Exec by sending logs to CloudWatch Logs using the `awslogs` log driver that's configured in your task definition. If you want to provide a custom configuration, the AWS CLI supports a `--configuration` flag for both the `create-cluster` and `update-cluster` commands. It's also important to know that the container image requires `script` and `cat` to be installed in order to have command logs uploaded correctly to Amazon S3 or CloudWatch Logs. For more information about creating clusters, see [create-cluster](#).

#### Note

This configuration only handles the logging of the `execute-command` session. It doesn't affect logging of your application.

The following example creates a service and then logs the output to your CloudWatch Logs LogGroup named `cloudwatch-log-group-name` and your Amazon S3 bucket named `s3-bucket-name`.

You must use an AWS KMS customer managed key to encrypt the log group when you set the `CloudWatchEncryptionEnabled` option to `true`. For information about how to encrypt the log group, see [Encrypt log data in CloudWatch Logs using AWS Key Management Service](#), in the *Amazon CloudWatch Logs User Guide*.

```
aws ecs create-cluster \
  --cluster-name cluster-name \
  --configuration executeCommandConfiguration="{ \
    kmsKeyId=string, \
    logging=OVERRIDE, \
    logConfiguration={ \
      cloudWatchLogGroupName=cloudwatch-log-group-name, \
      cloudWatchEncryptionEnabled=true, \
      s3BucketName=s3-bucket-name, \
      s3EncryptionEnabled=true, \
      s3KeyPrefix=demo \
    } \
  }"
```

The logging property determines the behavior of the logging capability of ECS Exec:

- `NONE`: logging is disabled

- **DEFAULT:** logs are sent to the configured `awslogs` driver (If the driver isn't configured, then no log is saved.)
- **OVERRIDE:** logs are sent to the provided Amazon CloudWatch Logs LogGroup, Amazon S3 bucket, or both

## IAM permissions required for Amazon CloudWatch Logs or Amazon S3 Logging

To enable logging, the Amazon ECS task role that's referenced in your task definition needs to have additional permissions. These additional permissions can be added as an inline policy to the task role. They are different depending on if you direct your logs to Amazon CloudWatch Logs or Amazon S3.

### Amazon CloudWatch Logs

The following example inline policy adds the required Amazon CloudWatch Logs permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:region:account-id:log-group:/aws/ecs/cloudwatch-log-group-name:"
    }
  ]
}
```

### Amazon S3

The following example inline policy adds the required Amazon S3 permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetEncryptionConfiguration"
      ],
      "Resource": "arn:aws:s3:::s3-bucket-name"
    }
  ],
}
```

```
{
  "Effect": "Allow",
  "Action": [
    "s3:PutObject"
  ],
  "Resource": "arn:aws:s3:::s3-bucket-name/*"
}
```

## IAM permissions required for encryption using your own AWS KMS key (KMS key)

By default, the data transferred between your local client and the container uses TLS 1.2 encryption that AWS provides. To further encrypt data using your own KMS key, you must create a KMS key and add the `kms:Decrypt` permission to your task IAM role. This permission is used by your container to decrypt the data. For more information about creating a KMS key, see [Creating keys](#).

You would add the following inlining policy to your task IAM role which requires the AWS KMS permissions. For more information, see [IAM permissions required for ECS Exec \(p. 435\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "kms-key-arn"
    }
  ]
}
```

For the data to be encrypted using your own KMS key, the user or group using the `execute-command` action must be granted the `kms:GenerateDataKey` permission.

The following example policy for your user or group contains the required permission to use your own KMS key. You must specify the Amazon Resource Name (ARN) of your KMS key.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey"
      ],
      "Resource": "kms-key-arn"
    }
  ]
}
```

## Using IAM policies to limit access to ECS Exec

You can limit user access to the `execute-command` API action by using one or more of the following IAM policy condition keys:



- `aws:ResourceTag/clusterTagKey`
- `ecs:ResourceTag/clusterTagKey`
- `aws:ResourceTag/taskTagKey`
- `ecs:ResourceTag/taskTagKey`
- `ecs:container-name`
- `ecs:cluster`
- `ecs:task`
- `ecs:enable-execute-command`

With the following example IAM policy, users can run commands in containers that are running within tasks with a tag that has an environment key and development value and in a cluster that's named cluster-name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:ExecuteCommand",
        "ecs:DescribeTasks"
      ],
      "Resource": "arn:aws:ecs:region:aws-account-id:task/cluster-name/*",
      "Condition": {
        "StringEquals": {
          "ecs:ResourceTag/environment": "development"
        }
      }
    }
  ]
}
```

With the following IAM policy example, users can't use the `execute-command` API when the container name is `production-app`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "ecs:ExecuteCommand"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "ecs:container-name": "production-app"
        }
      }
    }
  ]
}
```

With the following IAM policy, users can only launch tasks when ECS Exec is disabled.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Effect": "Allow",
  "Action": [
    "ecs:RunTask",
    "ecs:StartTask",
    "ecs:CreateService",
    "ecs:UpdateService"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "ecs:enable-execute-command": "false"
    }
  }
}
```

**Note**

Because the `execute-command` API action contains only task and cluster resources in a request, only cluster and task tags are evaluated.

For more information about IAM policy condition keys, see [Actions, resources, and condition keys for Amazon Elastic Container Service](#) in the *Service Authorization Reference*.

## Limiting access to the Start Session action

While starting SSM sessions on your container outside of ECS Exec is possible, this could potentially result in the sessions not being logged. Sessions started outside of ECS Exec also count against the session quota. We recommend limiting this access by denying the `ssm:start-session` action directly for your Amazon ECS tasks using an IAM policy. You can deny access to all Amazon ECS tasks or to specific tasks based on the tags used.

The following is an example IAM policy that denies access to the `ssm:start-session` action for tasks in all Regions with a specified cluster name. You can optionally include a wildcard with the `cluster-name`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "ssm:StartSession",
      "Resource": "arn:aws:ecs:*:111122223333:task/cluster-name/*"
    }
  ]
}
```

The following is an example IAM policy that denies access to the `ssm:start-session` action on resources in all Regions tagged with tag key `Task-Tag-Key` and tag value `Exec-Task`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "ssm:StartSession",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Task-Tag-Key": "Exec-Task"
        }
      }
    }
  ]
}
```

```
}  
  }  
    }  
      }  
        }
```

## Troubleshooting issues with ECS Exec

The following are troubleshooting notes to help diagnose why you may be getting an error when using ECS Exec.

### Verify using the Amazon ECS Exec Checker

The Amazon ECS Exec Checker script provides a way to verify and validate that your Amazon ECS cluster and task have met the prerequisites for using the ECS Exec feature. The Exec Checker script verifies both your AWS CLI environment and cluster and tasks are ready for ECS Exec, by calling various APIs on your behalf. The tool requires the latest version of the AWS CLI and that the `jq` is available. For more information, see [Amazon ECS Exec Checker](#) on GitHub.

### Error when calling `execute-command`

If a `The execute command failed` error occurs, the following are possible causes.

- The task does not have the required permissions. Verify that the task definition used to launch your task has a task IAM role defined and that the role has the required permissions. For more information, see [IAM permissions required for ECS Exec \(p. 435\)](#).
- The SSM agent is not installed or is not running
- There is an interface Amazon VPC endpoint for Amazon ECS, but there is not one for for Systems Manager Session Manager

## 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 stopped task errors like this in the Amazon ECS console by viewing the stopped task and inspecting it for error messages.

#### Important

Amazon ECS also sends task state change events to EventBridge, which you can view if your stopped task has expired from view in the Amazon ECS console. For more information, see [Task state change events \(p. 299\)](#).

For information about how to investigate a task that was stopped more than 1 hour, see [ECS Stopped Tasks in CloudWatch Logs](#) on the GitHub website.

New console

#### New AWS Management Console

The following steps can be used to set a container instance to draining using the new 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 the cluster.
4. On the **Cluster : *name*** page, choose the **Tasks** tab.

- Choose the stopped task to inspect.
- In the **Status** section, inspect the **Stopped reason** field to see the reason that the task was stopped.

#### Classic console

- Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
- On the **Clusters** page, select the cluster where your stopped task resides.
- On the **Cluster : *clustername*** page, choose **Tasks**.
- 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.
- In the **Details** section, inspect the **Stopped reason** field to see the reason that the task was stopped.

### Details

<b>Cluster</b>	default
<b>Container Instance</b>	dd3599e9-2ca6-40f4-9da5-a0bb10408260
<b>EC2 instance id</b>	i-83c6ab47
<b>Task Definition</b>	curler:4
<b>Last status</b>	STOPPED
<b>Desired status</b>	STOPPED
<b>Created at</b>	2015-11-20 13:31:01 -0800
<b>Stopped at</b>	2015-11-20 13:31:03 -0800
<b>Stopped reason</b>	Essential container in task exited

- If you have a container that has stopped, expand the container and inspect the **Status reason** row to see what caused the task state to change.

### Containers

Name	Container Id	Status
▼ curler	3f871451-c9f1-4d6f-a...	STOPPED (CannotPullContainerError: Error: image tutum/bogus)
<b>Details</b>		
<b>Status reason</b> CannotPullContainerError: Error: image tutum/bogus:latest not found <b>Command</b> ["/usr/bin/watch","curl","-v","http://amazon-ecs-2004772631.us-west-2.elb.amazonaws.com/"]		

In the previous example, the container image name can't be found. This can happen if you misspell the image name.

If this inspection doesn't provide enough information, see [Stopped tasks error codes \(p. 444\)](#) for more information.

#### AWS CLI

1. List the stopped tasks in a cluster. The output contains the Amazon Resource Name (ARN) of the task, which you need to describe the task.

```
aws ecs list-tasks \
  --cluster cluster_name \
  --desired-status STOPPED \
  --region us-west-2
```

2. Describe the stopped task to retrieve the `stoppedReason` in the response.

```
aws ecs describe-tasks \
  --cluster cluster_name \
  --tasks arn:aws:ecs:us-west-2:account_id:task/cluster_name/task_ID \
  --region us-west-2
```

## Additional resources

The following pages provide additional information about error codes:

- [Why is my Amazon ECS task stopped](#)
- [Stopped tasks error codes](#)

## Stopped tasks error codes

The following are the possible error messages you may receive when your Fargate task is stopped unexpectedly. The error messages are returned by the container agent and the prefix is dependent on the platform version the task is using.

To check your stopped tasks for an error message using the AWS Management Console, see [Checking stopped tasks for errors \(p. 442\)](#).

Error message prefix in platform version 1.3 and prior (Linux)	Error message prefix in platform version 1.4 and later (Linux) and version 1.0 and later (Windows)	Details	Example
DockerTimeoutError	ContainerRuntimeTimeoutError	This error occurs when a container can't transition to either a RUNNING or STOPPED state within the timeout period. The reason and timeout value is provided in the error message.	ContainerRuntimeTimeoutError: Could not transition to running; timed out after waiting 1m: <b>&lt;reason&gt;</b>

Error message prefix in platform version 1.3 and prior (Linux)	Error message prefix in platform version 1.4 and later (Linux) and version 1.0 and later (Windows)	Details	Example
CannotStartContainerError	CannotStartContainerError	This error occurs when a container can't be started.	CannotStartContainerError: failed to get container status: <b>&lt;reason&gt;</b>
CannotStopContainerError	CannotStopContainerError	This error occurs when a container can't be stopped.	CannotStopContainerError: failed sending SIGTERM to container: <b>&lt;reason&gt;</b>
CannotInspectContainerError	CannotInspectContainerError	<p>This error occurs when the container agent can't describe the container through the container runtime.</p> <p>When using platform version 1.3 or prior, the ECS agent will return the reason from Docker.</p> <p>When using platform version 1.4 or later 1.4.0 or later (Linux) or 1.0.0 or later (Windows), the Fargate agent will return the reason from containerd.</p>	CannotInspectContainerError: <b>&lt;reason&gt;</b>
	ResourceInitializationError	<p>This error occurs when the container agent for your Fargate task fails to create or bootstrap the resources required to start the container or the task is belongs to.</p> <p>A common cause for this error is using a VPC that doesn't have DNS resolution enabled.</p> <p>This error only occurs if you use platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows).</p>	ResourceInitializationError: failed to initialize logging driver: <b>&lt;reason&gt;</b>

Error message prefix in platform version 1.3 and prior (Linux)	Error message prefix in platform version 1.4 and later (Linux) and version 1.0 and later (Windows)	Details	Example
CannotPullContainerError	CannotPullContainerError	This error occurs when the agent is unable to pull the container image specified in the task definition. For more information, see <a href="#">CannotPullContainer task errors (p. 447)</a> .	CannotPullContainerError: <reason>
	CannotCreateVolumeError	This error occurs when the agent can't create the volume mount specified in the task definition.  This error only occurs if you use platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows).	CannotCreateVolumeError: <reason>
	ContainerRuntimeError	This error occurs when the agent receives an unexpected error from containerd for a runtime-specific operation. This error is usually caused by an internal failure in the agent or the containerd runtime.  This error only occurs if you use platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows).	ContainerRuntimeError: failed to create container IO set: <reason>
OutOfMemoryError	OutOfMemoryError	This error occurs when a container exits due to processes in the container consuming more memory than was allocated in the task definition.	OutOfMemoryError: container killed due to memory usage

Error message prefix in platform version 1.3 and prior (Linux)	Error message prefix in platform version 1.4 and later (Linux) and version 1.0 and later (Windows)	Details	Example
	InternalError	<p>This error occurs when the agent encounters an unexpected, non-runtime related internal error.</p> <p>This error only occurs if using platform version 1.4 or later.</p>	InternalError: <reason>
TaskFailedToStart	TaskFailedToStart	<p>This error occurs when an ENI attachment is requested. Amazon EC2 asynchronously handles the provisioning of the ENI. The provisioning process takes time. Amazon ECS has a timeout in case there are long wait times or unreported failures. There are times when the ENI is provisioned, but the report comes to Amazon ECS after the failure timeout. In this case, Amazon ECS sees the reported task failure with an in-use ENI.</p>	InternalError: <reason>
TaskFailedToStart	TaskFailedToStart	<p>This error occurs when the subnet that hosts the instances does not have enough IP addresses. The number of available IP addresses is available on the subnet details page, or by using <a href="#">describe-subnets</a>. For more information, see <a href="#">View your subnet</a> in the Amazon VPC User Guide.</p>	Unexpected EC2 error while attempting to Create Network Interface with public IP assignment enabled in subnet ' <a href="#">subnet-id</a> ': InsufficientFreeAddressesInSubnet

## CannotPullContainer task errors

The following errors indicate that, when creating a task, the container image specified can't be retrieved.



### Note

The 1.4 Fargate platform version truncates long error messages.

#### 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's because a route to the internet doesn't 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 [Run a standalone task \(p. 202\)](#).
- 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*.

#### Context canceled

The common cause for this error is because the VPC your task is using doesn't have a route to pull the container image from Amazon ECR.

#### Image not found

When you specify an Amazon ECR image in your container definition, you must use the full URI of your ECR repository along with the image name in that repository. If the repository or image can't 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 URI and the image name. Also make sure 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 \(p. 354\)](#).

#### 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, similar to the preceding example, the root volume might be filled out by a running container. If the container is using the default `json-file` log driver without a `max-size` limit, it may be that the log file is responsible for most of that space used. You can use the `docker ps` command to verify which container is using the space by mapping the directory name from the output above to the container ID. For example:

CONTAINER ID	IMAGE	COMMAND	CREATED
50501b5f4cbf	amazon/amazon-ecs-agent:latest	"/agent"	4 days ago
Up 4 days		ecs-agent	

By default, when using the `json-file` log driver, Docker captures the standard output (and standard error) of all of your containers and writes them in files using the JSON format. You can set the `max-size` as a log driver option, which prevents the log file from taking up too much space. For more information, see [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 \(p. 138\)](#).

## Docker Hub rate limiting

If you receive one of the following errors, you're likely hitting the Docker Hub rate limits:

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit. You may increase the limit by authenticating and
upgrading: https://www.docker.com/increase-rate-limits.
```

For more information about the Docker Hub rate limits, see [Understanding Docker Hub rate limiting](#).

If you have increased the Docker Hub rate limit and you need to authenticate your Docker pulls for your container instances, see [Private registry authentication for container instances](#) in the *Amazon Elastic Container Service Developer Guide*.

## Fail to copy image

If you receive an error similar to the following when launching a task, it's because there is no access to the image:

```
CannotPullContainerError: ref pull has been retried 1 time(s): failed to copy:
httpReaderSeeker: failed open: unexpected status code
```

To resolve this issue, you can:

- For Fargate tasks, see [How do I resolve the "cannotpullcontainererror" error for my Amazon ECS tasks on Fargate.](#)
- For other tasks, see [How do I resolve the "cannotpullcontainererror" error for my Amazon ECS tasks.](#)

For additional information about STOPPED errors, see [Stopped tasks error codes](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

## Service event messages

When troubleshooting a problem with a service, the first place you should check for diagnostic information is the service event log. You can view service events using the `DescribeServices` API, the AWS CLI, or by using the AWS Management Console.

When viewing service event messages using the Amazon ECS API, only the events from the service scheduler are returned. These include the most recent task placement and instance health events. However, the Amazon ECS console displays service events from the following sources.

- Task placement and instance health events from the Amazon ECS service scheduler. These events will have a prefix of **service** (*service-name*). To ensure that this event view is helpful, we only show the 100 most recent events and duplicate event messages are omitted until either the cause is resolved or six hours passes. If the cause is not resolved within six hours, you will receive another service event message for that cause.
- Service Auto Scaling events. These events will have a prefix of **Message**. The 10 most recent scaling events are shown. These events only occur when a service is configured with an Application Auto Scaling scaling policy.

Use the following steps to view your current service event messages.

### New 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 the cluster.
4. On the **Cluster : *name*** page, choose the **Services** tab.
5. Choose the service to inspect.
6. In the **Notifications** section, view the messages.

### Classic console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, select the cluster where 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.

## Details

<b>Cluster</b>	default
<b>Container Instance</b>	dd3599e9-2ca6-40f4-9da5-a0bb10408260
<b>EC2 instance id</b>	i-83c6ab47
<b>Task Definition</b>	curler:4
<b>Last status</b>	STOPPED
<b>Desired status</b>	STOPPED
<b>Created at</b>	2015-11-20 13:31:01 -0800
<b>Stopped at</b>	2015-11-20 13:31:03 -0800
<b>Stopped reason</b>	Essential container in task exited

- If you have a container that has stopped, expand the container and inspect the **Status reason** row to see what caused the task state to change.

## Containers

Name	Container Id	Status
▼ curler	3f871451-c9f1-4d6f-a...	STOPPED (CannotPullContainerError: Error: image tutum/bogus)
Details		
<b>Status reason</b> CannotPullContainerError: Error: image tutum/bogus:latest not found <b>Command</b> ["/usr/bin/watch","curl","-v","http://amazon-ecs-2004772631.us-west-2.elb.amazonaws.com/"]		

In the previous example, the container image name can't be found. This can happen if you misspell the image name.

If this inspection doesn't provide enough information, see [Stopped tasks error codes \(p. 444\)](#) for more information.

## AWS CLI

Use the [describe-services](#) command to view the service event messages for a specified service.

The following AWS CLI example describes the *service-name* service in the *default* cluster, which will provide the latest service event messages.

```
aws ecs describe-services \
  --cluster default \
  --services service-name \
  --region us-west-2
```

## Service event messages

The following are examples of service event messages you may see in the Amazon ECS console.

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. 274).

After the service is updated, for example with an updated task definition, the service scheduler resumes normal behavior.

service (*service-name*) operations are being throttled. Will try again later.

This service is unable to launch more tasks due to API throttling limits. Once the service scheduler is able to launch more tasks, it will resume.

To request an API rate limit quota increase, open the [AWS Support Center](#) page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.

service (*service-name*) was unable to stop or start tasks during a deployment because of the service deployment configuration. Update the `minimumHealthyPercent` or `maximumPercent` value and try again.

This service is unable to stop or start tasks during a service deployment due to the deployment configuration. The deployment configuration consists of the `minimumHealthyPercent` and `maximumPercent` values which are defined when the service is created, but can also be updated on an existing service.

The `minimumHealthyPercent` represents the lower limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for the service. This value is rounded up. For example if the minimum healthy percent is 50 and the desired task count is four, then the scheduler can stop two existing tasks before starting two new tasks. Likewise, if the minimum healthy percent is 75% and the desired task count is two, then the scheduler can't stop any tasks due to the resulting value also being two.

The `maximumPercent` represents the upper limit on the number of tasks that should be running for a service during a deployment or when a container instance is draining, as a percent of the desired number of tasks for a service. This value is rounded down. For example if the maximum percent is 200 and the desired task count is four then the scheduler can start four new tasks before stopping four existing tasks. Likewise, if the maximum percent is 125 and the desired task count is three, the scheduler can't start any tasks due to the resulting value also being three.

When setting a minimum healthy percent or a maximum percent, you should ensure that the scheduler can stop or start at least one task when a deployment is triggered.

service (*service-name*) was unable to place a task. Reason: You've reached the limit on the number of tasks you can run concurrently

You can request a quota increase for the resource that caused the error. For more information, see [the section called "Service quotas"](#) (p. 281). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

service (*service-name*) was unable to place a task. Reason: Internal error.

The service is unable to start a task due to a subnet being in an unsupported Availability Zone.

For information about the supported Fargate Regions and Availability Zones, see [the section called "AWS Fargate Regions"](#) (p. 285).

For information about how to view the subnet Availability Zone, see [View your subnet](#) in the *Amazon VPC User Guide*.

service (*service-name*) was unable to place a task. Reason: The requested CPU configuration is above your limit.

You can request a quota increase for the resource that caused the error. For more information, see [the section called "Service quotas"](#) (p. 281). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

service (*service-name*) was unable to place a task. Reason: The requested MEMORY configuration is above your limit.

You can request a quota increase for the resource that caused the error. For more information, see [the section called "Service quotas"](#) (p. 281). To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

## Invalid CPU or memory value specified

When registering a task definition using the Amazon ECS API or AWS CLI, if you specify an invalid `cpu` or memory value, the following error is returned.

An error occurred (ClientException) when calling the RegisterTaskDefinition operation: Invalid 'cpu' setting for task. For more information, see the Troubleshooting section of the Amazon ECS Developer Guide.

### Note

When using Terraform, the following error may be returned.

```
Error: ClientException: No Fargate configuration exists for given values.
```

To resolve this issue, you must specify a supported value for the task CPU and memory in your task definition. The `cpu` value can be expressed in CPU units or vCPUs in a task definition but is converted to an integer indicating the CPU units when the task definition is registered. The `memory` value can be expressed in MiB or GB in a task definition but is converted to an integer indicating the MiB when the task definition is registered.

For task definitions that only specify `EC2` for the `requiresCompatibilities` parameter, the supported CPU values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs). The memory value must be an integer and the limit is dependent upon the amount of available memory on the underlying Amazon EC2 instance you use.

For task definitions that specify `FARGATE` for the `requiresCompatibilities` parameter (even if `EC2` is also specified), you must use one of the values in the following table, which determines your range of supported values for the CPU and memory parameter.

Supported task CPU and memory values for tasks that are hosted on Fargate 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)

## 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 aren't 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.

#### Container instance security group

If your container is mapped to port 80 on your container instance, your container instance security group must allow inbound traffic on port 80 for the load balancer health checks to pass.

#### Elastic Load Balancing load balancer not configured for all Availability Zones

Your load balancer should be configured to use all of the Availability Zones in a Region, or at least all of the Availability Zones where your container instances reside. If a service uses a load balancer and starts a task on a container instance that resides in an Availability Zone that the load balancer isn't configured to use, the task never passes the health check and it's killed.

#### Elastic Load Balancing load balancer health check misconfigured

The load balancer health check parameters can be overly restrictive or point to resources that don't exist. If a container instance is determined to be unhealthy, it 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're using with the health check.

##### Ping Path

This value is often set to `index.html`, but if your service doesn't respond to that request, then the health check fails. If your container doesn't have an `index.html` file, you can set this to `/` to target the base URL for the container instance.

##### Response Timeout

This is the amount of time that your container has to return a response to the health check ping. If this value is lower than the amount of time required for a response, the health check fails.

#### Health Check Interval

This is the amount of time between health check pings. The shorter your health check intervals are, the faster your container instance can reach the **Unhealthy Threshold**.

#### Unhealthy Threshold

This is the number of times your health check can fail before your container instance is considered unhealthy. If you have an unhealthy threshold of 2, and a health check interval of 30 seconds, then your task has 60 seconds to respond to the health check ping before it 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.

You've reached the limit on the number of tasks you can run concurrently.

For a new account, your quotas might be lower than the service quotas. The service quota for your account can be viewed in the Service Quotas console. To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

## Troubleshooting service auto scaling

Application Auto Scaling disables scale-in processes while Amazon ECS deployments are in progress and they resume once the deployment has completed. However, scale-out processes continue to occur, unless suspended, during a deployment. For more information, see [Suspending and resuming scaling for Application Auto Scaling](#).

## AWS Fargate throttling quotas

AWS Fargate limits Amazon ECS tasks and Amazon EKS pods launch rates to quotas (formerly referred to as limits) using a [token bucket algorithm](#) for each AWS account on a per-Region basis. With this algorithm, your account has a bucket that holds a specific number of tokens. The number of tokens in the bucket represents your rate quota at any given second. Each customer account has a tasks and pods token bucket that depletes based on the number of tasks and pods launched by the customer account. This token bucket has a bucket maximum that allows you to periodically make a higher number of requests, and a refill rate that allows you to sustain a steady rate of requests for as long as needed.

For example, the tasks and pods token bucket size for a Fargate customer account is 100 tokens, and the refill rate is 20 tokens per second. Therefore, you can immediately launch up to 100 Amazon ECS tasks and Amazon EKS pods per customer account, with a sustained launch rate of 20 Amazon ECS tasks and Amazon EKS pods per second.

Actions	Bucket maximum capacity (or Burst rate)	Bucket refill rate (or Sustained rate)
Fargate Resource rate quota for On Demand Amazon ECS tasks and Amazon EKS pods <sup>1</sup> (p. 456)	100	20



Actions	Bucket maximum capacity (or Burst rate)	Bucket refill rate (or Sustained rate)
Fargate Resource rate quota for Spot Amazon ECS tasks	100	20

<sup>1</sup>Accounts launching only Amazon EKS pods have a burst rate of 20 with a sustained pod launch rate of 20 pod launches per second when using the platform versions called out in the [Amazon EKS platform versions](#).

## Throttling the RunTask API

In addition, Fargate limits the request rate when launching tasks using the Amazon ECS RunTask API using a separate quota. Fargate limits Amazon ECS RunTask API requests for each AWS account on a per-Region basis. Each request that you make removes one token from the bucket. We do this to help the performance of the service, and to ensure fair usage for all Fargate customers. API calls are subject to the request quotas whether they originate from the Amazon Elastic Container Service console, a command line tool, or a third-party application. The rate quota for calls to the Amazon ECS RunTask API is 20 calls per second (burst and sustained). Each call to this API can, however, launch up to 10 tasks. This means you can launch 100 tasks in one second by making 10 calls to this API, requesting 10 tasks to be launched in each call. Similarly, you could also make 20 calls to this API, requesting 5 tasks to be launched in each call. For more information on API throttling for Amazon ECS RunTask API, see [API request throttling](#) in the Amazon ECS API Reference.

In practice, task and pod launch rates are also dependent on other considerations such as container images to be downloaded and unpacked, health checks and other integrations enabled, such as registering tasks or pods into a load balancer. Customers will see variations in task and pod launch rates compared with the quotas represented above based on the features that customers enable.

## Adjusting rate quotas

You can request an increase for Fargate rate throttling quotas for your AWS account. To request a quota adjustment, contact the [AWS Support Center](#).

## API failure reasons

When an API action that you have triggered through the Amazon ECS API, console, or the AWS CLI exits with a `failures` error message, the following may assist in troubleshooting the cause. The failure will return a reason and the Amazon Resource Name (ARN) of the resource associated with the failure.

Many resources are Region-specific, so when using the console ensure that you set the correct Region for your resources. When using the AWS CLI, make sure that your AWS CLI commands are being sent to the correct Region with the `--region region` parameter.

For more information about the structure of the `Failure` data type, see [Failure](#) in the *Amazon Elastic Container Service API Reference*.

The following are examples of failure messages you might receive when running API commands.

API action	Failure reason	Cause
<code>DescribeClusters</code>	<code>MISSING</code>	The specified cluster wasn't found. Verify the spelling of the cluster name.

API action	Failure reason	Cause
DescribeServices	MISSING	The specified service wasn't found. Verify that the correct cluster or Region is specified and that the service ARN or name is valid.
DescribeTasks	MISSING	The specified task wasn't found. Verify the correct cluster or Region is specified and that both the task ARN or ID is valid.

# Document history

The following table describes the major updates and new features for the *Amazon ECS User Guide for AWS Fargate*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
Support for Amazon ECS Exec on Fargate Windows containers	Amazon ECS Exec supports Fargate Windows containers. For more information, see <a href="#">Using Amazon ECS Exec for debugging (p. 433)</a> .	
The new Amazon ECS console experience updated	The new Amazon ECS console experience supports creating and deleting a cluster, updating a task definition, and deregistering a task definition. For more information, see <a href="#">Creating a cluster for the Fargate launch type using the new console (p. 63)</a> , <a href="#">Deleting a cluster using the new console (p. 65)</a> , <a href="#">Updating a task definition using the new console (p. 69)</a> , and <a href="#">Deregistering a task definition revision (p. 194)</a> .	08-Dec 2021
Amazon ECS support for the fluentd log-driver-buffer-limit option	Amazon ECS supports the <code>fluentd log-driver-buffer-limit</code> option. For more information, see <a href="#">Custom log routing (p. 144)</a> .	23 Nov 2021
The new Amazon ECS console experience updated	The new Amazon ECS console experience supports creating a task definition. For more information, see <a href="#">Creating a task definition using the new console (p. 66)</a> .	23 Nov 2021
Amazon ECS supports the 64-bit ARM architecture for Linux.	Amazon ECS supports the 64-bit ARM CPU architecture for the Linux operating system. For more information, see <a href="#">the section called "Working with 64-bit ARM workloads on Amazon ECS" (p. 125)</a> .	23 Nov 2021
Support for Windows Amazon ECS Exec	Amazon ECS Exec supports Windows. For more information, see <a href="#">Using Amazon ECS Exec for debugging (p. 433)</a> .	01 Nov 2021
Support for Windows on Fargate.	Amazon ECS supports Windows on Fargate. For more information, see <a href="#">What is AWS Fargate? (p. 1)</a> .	28 October 2021
Amazon ECS metadata update	Added support for the container clock diff information in the metadata endpoint version 4. For more information, see <a href="#">the section called "Task metadata JSON response" (p. 381)</a> .	30 September 2021
Amazon ECS scheduled tasks update	Amazon EventBridge added support for additional parameters when creating rules that trigger Amazon ECS scheduled tasks. For more information, see <a href="#">Scheduled tasks (p. 204)</a> .	25 June 2021
Getting started with the AWS CDK	Added a getting started guide for using the AWS CDK with Amazon ECS. For more information, see <a href="#">Getting started with Amazon ECS using the AWS CDK (p. 16)</a> .	27 May 2021
Getting started with the AWS CDK	Added a getting started guide for using the AWS CDK with Amazon ECS. For more information, see <a href="#">Getting started with Amazon ECS using the AWS CDK (p. 16)</a> .	27 May 2021

Change	Description	Date
Amazon ECS Exec	Amazon ECS has released a new debugging tool called ECS Exec. For more information, see <a href="#">Using Amazon ECS Exec for debugging (p. 433)</a> .	15 March 2021
New console experience	Amazon ECS has released a new console experience which supports creating or updating a service or running a standalone task. For more information, see <a href="#">Creating a service using the new console (p. 72)</a> and <a href="#">Run a standalone task (p. 202)</a> .	28 December 2020
AWS Fargate platform version 1.4.0 update	<p>Beginning on November 5, 2020, any new Fargate task that is launched using platform version 1.4.0 will have access to the following features. For more information, see <a href="#">1.4.0 (p. 58)</a>.</p> <ul style="list-style-type: none"> <li>• When using Secrets Manager to store sensitive data, you can inject a specific JSON key or a specific version of a secret as an environment variable or in a log configuration. For more information, see <a href="#">Specifying sensitive data using Secrets Manager (p. 176)</a>.</li> <li>• Specify environment variables in bulk using the <code>environmentFiles</code> container definition parameter. For more information, see <a href="#">Specifying environment variables (p. 186)</a>.</li> <li>• Tasks run in a VPC and subnet enabled for IPv6 will be assigned both a private IPv4 address and an IPv6 address. For more information, see <a href="#">Fargate task networking</a> in the <i>Amazon Elastic Container Service User Guide for AWS Fargate</i>.</li> <li>• The task metadata endpoint version 4 provides additional metadata about your task and container including the task launch type, the Amazon Resource Name (ARN) of the container, and the log driver and log driver options used. When querying the <code>/stats</code> endpoint you also receive network rate stats for your containers. For more information, see <a href="#">Task metadata endpoint version 4</a> in the <i>Amazon Elastic Container Service User Guide for AWS Fargate</i>.</li> </ul>	5 November 2020
Fargate usage metrics	AWS Fargate provides CloudWatch usage metrics which provide visibility into your accounts usage of Fargate On-Demand and Fargate Spot resources. For more information, see <a href="#">AWS Fargate usage metrics (p. 296)</a> .	3 August 2020
AWS Copilot version 0.1.0	The new AWS Copilot CLI launched, providing high-level commands to simplify modeling, creating, releasing, and managing containerized applications on Amazon ECS from a local development environment. For more information, see <a href="#">Using the AWS Copilot command line interface (p. 35)</a> .	9 July 2020
AWS Fargate platform versions deprecation schedule	The Fargate platform version deprecation schedule has been added. For more information, see <a href="#">AWS Fargate platform version deprecation (p. 61)</a> .	8 July 2020

Change	Description	Date
AWS Fargate Region expansion	Amazon ECS on AWS Fargate has expanded to the Europe (Milan) Region.	25 June 2020
AWS Fargate platform version 1.4.0 update	Beginning on May 28, 2020, any new Fargate task that is launched using platform version 1.4.0 will have its 20 GB ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS Fargate-managed encryption key. For more information, see <a href="#">Using data volumes in tasks (p. 127)</a> .	28 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	<p>The following service quota was updated:</p> <ul style="list-style-type: none"><li>• Clusters per account was raised from 2,000 to 10,000.</li></ul> <p>For more information, see <a href="#">Amazon ECS service quotas (p. 281)</a>.</p>	17 April 2020

Change	Description	Date
AWS Fargate platform version 1.4.0	<p>AWS Fargate platform version 1.4.0 is released, which contains the following features:</p> <ul style="list-style-type: none"> <li>Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see <a href="#">Amazon EFS volumes (p. 129)</a>.</li> <li>The ephemeral task storage has been increased to 20 GB. For more information, see <a href="#">Using data volumes in tasks (p. 127)</a>.</li> <li>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. 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 <a href="#">Fargate task networking (p. 136)</a>.</li> <li>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 larger the application payload that can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames reduces overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC.</li> <li>CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see <a href="#">Amazon ECS CloudWatch Container Insights (p. 308)</a>.</li> <li>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 <a href="#">Task metadata endpoint version 4 (p. 380)</a>.</li> <li>Added support for the <code>SYS_PTRACE</code> Linux parameter in container definitions. For more information, see <a href="#">Linux parameters (p. 115)</a>.</li> <li>The Fargate container agent replaces the use of the Amazon ECS container agent for all Fargate tasks. This change should not affect how your tasks run.</li> <li>The container runtime is now using Containerd instead of Docker. This change should not affect how your tasks run. You may notice that some error messages that originate with the container runtime are more general and do not mention Docker.</li> </ul> <p>For more information, see <a href="#">AWS Fargate platform versions (p. 58)</a>.</p>	8 April 2020

Change	Description	Date
Amazon EFS file system support for task volumes	Amazon EFS file systems can be used as data volumes for your Fargate tasks. For more information, see <a href="#">Amazon EFS volumes</a> (p. 129).	8 April 2020
Amazon ECS Task Metadata Endpoint version 4	Beginning with Fargate platform version 1.4.0, an environment variable named <code>ECS_CONTAINER_METADATA_URI_V4</code> is injected into each container in a task. When you query the task metadata version 4 endpoint, various task metadata and <a href="#">Docker stats</a> are available to tasks. For more information, see <a href="#">Task metadata endpoint version 4</a> (p. 380).	8 April 2020
Fargate Spot	Amazon ECS added support for running tasks using Fargate Spot. For more information, see <a href="#">AWS Fargate capacity providers</a> (p. 79).	3 Dec 2019
Service Action Events	Amazon ECS now sends events to Amazon EventBridge when certain service actions occur. For more information, see <a href="#">Service action events</a> (p. 301).	25 Nov 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 <a href="#">Custom log routing</a> (p. 144).	30 Sept 2019
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
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 <a href="#">Custom log routing</a> (p. 144).	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 <a href="#">Amazon ECS CloudWatch Container Insights</a> (p. 308).	30 Aug 2019
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Hong Kong) Region.	06 Aug 2019
Registering Multiple Target Groups with a Service	Added support for specifying multiple target groups in a service definition. For more information, see <a href="#">Registering multiple target groups with a service</a> (p. 261).	30 July 2019
CloudWatch Container Insights	Amazon ECS has added support for CloudWatch Container Insights. For more information, see <a href="#">Amazon ECS CloudWatch Container Insights</a> (p. 308).	9 July 2019
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 <a href="#">How Amazon Elastic Container Service works with IAM</a> (p. 325).	27 June 2019

Change	Description	Date
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports the <code>splunk</code> log driver in addition to the <code>awslogs</code> log driver. For more information, see <a href="#">Storage and logging (p. 109)</a> .	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 <code>secretOptions</code> container definition parameter. For more information, see <a href="#">Specifying sensitive data (p. 176)</a> .	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 <a href="#">Specifying sensitive data (p. 176)</a> .	2 Apr 2019
AWS Fargate platform version 1.3.0 update	Beginning on March 27, 2019, any new Fargate task 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 <a href="#">Proxy configuration (p. 119)</a> , <a href="#">Container dependency (p. 116)</a> , and <a href="#">Container timeouts (p. 117)</a> .	27 Mar 2019
Amazon ECS introduces the external deployment type	The <i>external</i> 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 <a href="#">External deployment (p. 247)</a> .	27 Mar 2019
Amazon ECS introduces the <code>PutAccountSettingDefault</code> API	Amazon ECS introduces the <code>PutAccountSettingDefault</code> 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 <a href="#">Amazon Resource Names (ARNs) and IDs (p. 197)</a> .	8 Feb 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 <a href="#">Interface VPC Endpoints (AWS PrivateLink)</a> .  26 Dec 2018	



Change	Description	Date
AWS Fargate platform version 1.3.0	<p>New AWS Fargate platform version released, which contains:</p> <ul style="list-style-type: none"> <li>Added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers.</li> </ul> <p>For more information, see <a href="#">Specifying sensitive data (p. 176)</a>.</p> <ul style="list-style-type: none"> <li>Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service.</li> </ul> <p>For more information, <a href="#">Task maintenance</a> in the <i>Amazon Elastic Container Service User Guide for AWS Fargate</i>.</p> <p>For more information, see <a href="#">AWS Fargate platform versions (p. 58)</a>.</p>	17 Dec 2018
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Mumbai) and Canada (Central) Regions.	07 Dec 2018
Amazon ECS blue/green deployments	<p>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.</p> <p>For more information, see <a href="#">Blue/Green deployment with CodeDeploy (p. 243)</a>.</p>	27 Nov 2018
Resource tagging	<p>Amazon ECS added support for adding metadata tags to your services, task definitions, tasks, clusters, and container instances.</p> <p>For more information, see <a href="#">Resources and tags (p. 276)</a>.</p>	15 Nov 2018
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the US West (N. California) and Asia Pacific (Seoul) Regions.</p> <p>For more information, see <a href="#">AWS Fargate platform versions (p. 58)</a>.</p>	07 Nov 2018
Service limits updated	<p>The following service limits were updated:</p> <ul style="list-style-type: none"> <li>Number of tasks using the Fargate launch type, per Region, per account was raised from 20 to 50.</li> <li>Number of public IP addresses for tasks using the Fargate launch type was raised from 20 to 50.</li> </ul> <p>For more information, see <a href="#">Amazon ECS service quotas (p. 281)</a>.</p>	31 Oct 2018

Change	Description	Date
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the Europe (London) Region.</p> <p>For more information, see <a href="#">AWS Fargate platform versions (p. 58)</a>.</p>	26 Oct 2018
Private registry authentication support for Amazon ECS using AWS Fargate tasks	<p>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.</p> <p>For more information, see <a href="#">Private registry authentication for tasks (p. 174)</a>.</p>	10 Sept 2018
Amazon ECS CLI v1.8.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> <li>• Added support for Docker volumes in Docker compose files.</li> <li>• Added support for task placement constraints and strategies in Docker compose files.</li> <li>• Added support for private registry authentication in Docker compose files.</li> <li>• Added support for <code>--force-update on compose up</code> to force relaunching of tasks.</li> </ul> <p>For more information, see the <a href="#">Amazon ECS Command Line Reference</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</p>	7 Sept 2018
Amazon ECS service discovery Region expansion	<p>Amazon ECS service discovery has expanded support to the Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), EU (Frankfurt), and Europe (London) Regions.</p> <p>For more information, see <a href="#">Service Discovery (p. 271)</a>.</p>	30 August 2018
Scheduled tasks with Fargate tasks support	<p>Amazon ECS introduced support for scheduled tasks for the Fargate launch type.</p> <p>For more information, see <a href="#">Scheduled tasks (p. 204)</a>.</p>	28 August 2018
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the Europe (Frankfurt), Asia Pacific (Singapore), and Asia Pacific (Sydney) Regions.</p> <p>For more information, see <a href="#">AWS Fargate platform versions (p. 58)</a>.</p>	19 July 2018

Change	Description	Date
Amazon ECS CLI v1.7.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> <li>Added support for container <code>healthcheck</code> and <code>devices</code> in Docker compose files. For more information, see the <a href="#">Amazon ECS Command Line Reference</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</li> </ul>	18 July 2018
Amazon ECS service scheduler strategies added	<p>Amazon ECS introduced the concept of service scheduler strategies.</p> <p>There are two service scheduler strategies available:</p> <ul style="list-style-type: none"> <li><b>REPLICA</b>—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 <a href="#">Replica (p. 213)</a>.</li> <li><b>DAEMON</b>—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 <a href="#">Daemon</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</li> </ul> <p><b>Note</b> Fargate tasks do not support the <b>DAEMON</b> scheduling strategy.</p> <p>For more information, see <a href="#">Service scheduler concepts (p. 212)</a>.</p>	12 June 2018
Amazon ECS CLI v1.6.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> <li>Added support for <a href="#">Docker compose file syntax</a> version 3. For more information, see the <a href="#">Amazon ECS Command Line Reference</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</li> </ul>	5 June 2018
AWS Fargate Region expansion	<p>AWS Fargate with Amazon ECS has expanded to the US East (Ohio), US West (Oregon), and EU West (Ireland) Regions.</p> <p>For more information, see <a href="#">AWS Fargate platform versions (p. 58)</a>.</p>	26 April 2018

Change	Description	Date
Amazon ECS CLI v1.5.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> <li>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.</li> <li>Added support for the <code>shm_size</code> and <code>tmpfs</code> parameters in compose files.</li> </ul> <p>For more information, see the <a href="#">Amazon ECS Command Line Reference</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</p>	19 April 2018
Amazon ECS CLI download verification	<p>Added new PGP signature method for verifying the Amazon ECS CLI installation file. For more information, see <a href="#">Installing the Amazon ECS CLI (p. 41)</a>.</p>	5 April 2018
AWS Fargate Platform Version	<p>New AWS Fargate platform version released, which contains:</p> <ul style="list-style-type: none"> <li>Added support for <a href="#">Amazon ECS task metadata endpoint (p. 380)</a>.</li> <li>Added support for <a href="#">Health check (p. 104)</a>.</li> <li>Added support for <a href="#">Service Discovery (p. 271)</a></li> </ul> <p>For more information, see <a href="#">AWS Fargate platform versions (p. 58)</a>.</p>	26 March 2018
Amazon ECS Service Discovery	<p>Added integration with Route 53 to support Amazon ECS service discovery. For more information, see <a href="#">Service Discovery (p. 271)</a>.</p>	22 March 2018
Amazon ECS CLI v1.4.2	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> <li>Updated the AMI to <code>amzn-ami-2017.09.k-amazon-ecs-optimized</code>.</li> </ul> <p>For more information, see the <a href="#">Amazon ECS Command Line Reference</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</p>	20 March 2018

Change	Description	Date
Amazon ECS CLI v1.4.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> <li>Added support for the us-gov-west-1 Region.</li> <li>Added <code>--force-deployment</code> flag for the <code>compose service</code> command.</li> <li>Added support for <code>aws_session_token</code> in ECS profiles.</li> <li>Updated the AMI to <code>amzn-ami-2017.09.j-amazon-ecs-optimized</code>.</li> </ul> <p>For more information, see the <a href="#">Amazon ECS Command Line Reference</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</p>	09 March 2018
Container Health Checks	<p>Added support for Docker health checks in container definitions. For more information, see <a href="#">Health check (p. 104)</a>.</p>	08 March 2018
Amazon ECS Task Metadata Endpoint	<p>Beginning with version 1.17.0 of the Amazon ECS container agent, various task metadata and <a href="#">Docker stats</a> are available to tasks that use the <code>awsvpc</code> network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see <a href="#">Amazon ECS task metadata endpoint (p. 380)</a>.</p>	8 February 2018
Amazon ECS Service Auto Scaling using target tracking policies	<p>Added support for ECS Service Auto Scaling using target tracking policies in the Amazon ECS console. For more information, see <a href="#">Target tracking scaling policies (p. 265)</a>.</p> <p>Removed the previous tutorial for step scaling in the ECS first run wizard. This was replaced with the new tutorial for target tracking.</p>	8 February 2018
Amazon ECS CLI v1.3.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> <li>Ability to create empty clusters with the <code>up</code> command.</li> <li>Added <code>--health-check-grace-period</code> flag for the <code>compose service up</code> command.</li> <li>Updated the AMI to <code>amzn-ami-2017.09.g-amazon-ecs-optimized</code>.</li> </ul> <p>For more information, see the <a href="#">Amazon ECS Command Line Reference</a> in the <i>Amazon Elastic Container Service Developer Guide</i>.</p>	19 January 2018
New service scheduler behavior	<p>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 <a href="#">Additional service concepts (p. 213)</a>.</p>	11 January 2018

Change	Description	Date
Task-level CPU and memory	Added support for specifying CPU and memory at the task-level in task definitions. For more information, see <a href="#">TaskDefinition</a> .	12 December 2017
Task execution role	<p>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:</p> <ul style="list-style-type: none"><li>• Calls to Amazon ECR to pull the container image</li><li>• Calls to CloudWatch to store container application logs</li></ul> <p>For more information, see <a href="#">Amazon ECS task execution IAM role (p. 354)</a>.</p>	7 December 2017
Amazon ECS CLI v1.1.0 with Fargate support	<p>New version of the Amazon ECS CLI released, which added the following features:</p> <ul style="list-style-type: none"><li>• Support for task networking</li><li>• Support for AWS Fargate</li><li>• Support for viewing CloudWatch Logs data from a task</li></ul> <p>For more information, see <a href="#">ECS CLI changelog</a>.</p>	29 November 2017
AWS Fargate GA	Added support for launching Amazon ECS services using the Fargate launch type. For more information, see <a href="#">Amazon ECS launch types (p. 124)</a> .	29 November 2017

# AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.