
Amazon Elastic Container Service

Developer Guide



Amazon Elastic Container Service: Developer Guide

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 Amazon ECS?	1
Features of Amazon ECS	1
Containers and images	2
Task definitions	3
Tasks and scheduling	4
Clusters	4
Container agent	4
Getting started with Amazon ECS	5
Related services	5
Pricing	6
Setting up	7
Sign up for AWS	7
Create an IAM user	7
Create a key pair	9
Create a virtual private cloud	11
Create a security group	12
Install the AWS CLI	13
Getting started	14
Docker basics	14
Installing Docker	14
Create a Docker image	15
Push your image to Amazon Elastic Container Registry	17
Clean up	18
Using AWS Copilot	18
Prerequisites	18
Deploy your application using one command	19
Deploy your application step by step	19
Using the AWS CDK	23
Step 1: Set up your AWS CDK project	23
Step 2: Use the AWS CDK to define a containerized Web server on Fargate	25
Step 3: Test the Web server	29
Step 4: Clean up	29
Next steps	29
Getting started with the Amazon ECS console using Linux containers on AWS Fargate	30
Prerequisites	30
Step 1: Create a task definition	31
Step 2: Configure the service	31
Step 3: Configure the cluster	32
Step 4: Review	32
Step 5: View your service	32
Step 6: Clean up	33
Getting started with the Amazon ECS console using Windows containers on AWS Fargate	33
Prerequisites	30
Step 1: Create a cluster	34
Step 2: Register a Windows task definition	34
Step 3: Create a service with your task definition	36
Step 4: View your service	36
Step 5: Clean Up	37
Getting started with the Amazon ECS console using Amazon EC2	37
Prerequisites	38
Step 1: Register a task definition	38
Step 2: Create a cluster	39
Step 3: Create a Service	39
Step 4: View your Service	40

Step 5: Clean Up	41
Using the console with Amazon EC2 Windows containers	41
Step 1: Create a Windows cluster	42
Step 2: Register a Windows task definition	43
Step 3: Create a service with your task definition	44
Step 4: View your service	45
Developer tools overview	46
AWS Management Console	46
AWS Command Line Interface	46
AWS CloudFormation	47
AWS Copilot CLI	47
AWS CDK	47
AWS App2Container	48
Amazon ECS CLI	48
Docker Desktop integration with Amazon ECS	48
AWS SDKs	49
Summary	49
Using the AWS Copilot CLI	49
Installing the AWS Copilot CLI	50
Next steps	54
Using the Amazon ECS CLI	54
Installing the Amazon ECS CLI	55
Configuring the Amazon ECS CLI	60
Migrating Configuration Files	61
Tutorial: Creating a cluster with a Fargate task using the Amazon ECS CLI	62
Tutorial: Creating a Cluster with an EC2 Task Using the Amazon ECS CLI	67
Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI	70
Amazon ECS command line reference	73
AWS Fargate	161
Task definitions	161
Network mode	162
Task Operating Systems	162
Task CPU and memory	162
Task resource limits	163
Logging	163
Amazon ECS task execution IAM role	163
Example Amazon Linux 2 task definition	164
Example Windows task definition	164
Task storage	165
Tasks and services	165
Task networking	165
Service load balancing	166
Private registry authentication	166
Clusters	167
Fargate Spot	167
Usage metrics	167
Task maintenance	167
Savings plans	168
Windows containers on Fargate considerations	168
Platform Versions	169
Linux platform versions	169
Windows platform versions	173
Getting started walkthroughs	174
Clusters	175
Cluster concepts	175
Creating a cluster	176

Capacity providers	178
Capacity provider concepts	179
Capacity provider considerations	179
AWS Fargate capacity providers	180
Auto Scaling group capacity providers	184
Cluster auto scaling	189
Cluster auto scaling considerations	190
Managed scale-out behavior	190
Using Local Zones, Wavelength Zones, and AWS Outposts	191
Local Zones	191
Wavelength Zones	191
AWS Outposts	192
Updating cluster settings	192
Deleting a cluster	192
Task definitions	194
Application architecture	195
Using the Fargate launch type	195
Using the EC2 launch type	195
Creating a task definition using the new console	196
Creating a task definition using the classic console	198
Task definition template	205
Task definition parameters	209
Family	209
Platform family	209
Launch types	210
Task role	210
Task execution role	210
Network mode	211
Runtime platform	211
Task size	212
Container definitions	213
Task placement constraints	239
Proxy configuration	240
Volumes	241
Tags	245
Other task definition parameters	246
Launch types	247
Fargate launch type	247
EC2 launch type	248
External launch type	249
Working with GPUs on Amazon ECS	250
Considerations	251
Specifying GPUs in your task definition	252
Working with inference workloads on Amazon ECS	253
Considerations	253
Using the Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI	253
Task definition requirements	254
Using data volumes in tasks	256
Fargate task storage	256
Amazon EFS volumes	257
FSx for Windows File Server volumes	260
Docker volumes	263
Bind mounts	268
Managing container swap space	277
Container swap considerations	278
Task networking	278
AWSVPC mode	278

Bridge mode	282
Host mode	283
Using the awslogs log driver	283
Enabling the awslogs log driver for your containers	283
Creating a log group	284
Available awslogs log driver options	284
Specifying a log configuration in your task definition	286
Viewing awslogs container logs in CloudWatch Logs	287
Custom log routing	289
Considerations	289
Required IAM permissions	290
Using Fluent logger libraries or Log4j over TCP	291
Using the AWS for Fluent Bit image	291
Creating a task definition that uses a FireLens configuration	292
Filtering logs using regular expressions	295
Example task definitions	296
Private registry authentication for tasks	300
Required IAM permissions for private registry authentication	301
Enabling private registry authentication	302
Specifying sensitive data	303
Using Secrets Manager	303
Using Systems Manager Parameter Store	310
Specifying environment variables	315
Considerations for specifying environment variable files	316
Required IAM permissions	316
Example task definitions	317
Example: Webserver	317
Example: splunk log driver	319
Example: fluentd log driver	319
Example: gelf log driver	320
Example: Amazon ECR image and task definition IAM role	320
Example: Entrypoint with command	321
Example: Container dependency	321
Updating a task definition	322
Deregistering a task definition revision	323
Account settings	324
Amazon Resource Names (ARNs) and IDs	325
ARN and resource ID format timeline	326
Viewing account settings	327
Modifying account settings	328
Container instances	330
Container instance concepts	330
Container instance lifecycle	331
Check the instance IAM role for your account	332
Linux instances	332
Amazon ECS-optimized AMI	333
Bottlerocket	362
Launching a container instance	364
Bootstrap Container Instances	368
Starting a task at container instance launch time	369
Elastic network interface trunking	372
Memory Management	382
Connect to your container instance	384
Manage container instances remotely	385
Windows instances	387
Amazon ECS-optimized AMI	387
Launching a container instance	408

Bootstrap Container Instances	411
Connect to your container Windows instance	413
External instances	414
Considerations	415
IAM permissions	416
Registering an external instance to a cluster	418
Deregistering an external instance	421
Running workloads on external instances	422
Updating the AWS Systems Manager Agent and Amazon ECS container agent	423
Monitoring	426
CloudWatch Logs IAM Policy	426
Installing and configuring the CloudWatch agent	427
Viewing CloudWatch Logs	427
Container instance draining	428
Draining container instances	429
Deregister a container instance	429
Container agent	431
Installing the Amazon ECS container agent	431
Installing the Amazon ECS container agent on an Amazon Linux 2 EC2 instance	432
Installing the Amazon ECS container agent on an Amazon Linux AMI EC2 instance	432
Installing the Amazon ECS container agent on a non-Amazon Linux EC2 instance	433
Running the Amazon ECS Container Agent with Host Network Mode	440
Container agent versions	440
Amazon ECS-Optimized Amazon Linux 2 AMI Container Agent Versions	441
Amazon ECS-Optimized Amazon Linux AMI Container Agent Versions	444
Updating the Amazon ECS container agent	448
Checking the Amazon ECS container agent version	448
Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI	449
Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs)	452
Container agent configuration	454
Available Parameters	454
Storing Container Instance Configuration in Amazon S3	467
Private registry authentication for container instances	468
Authentication formats	468
Enabling private registries	470
Automated task and image cleanup	471
Tunable parameters	472
Cleanup workflow	472
Container metadata file	472
Enabling container metadata	473
Container metadata file locations	473
Container metadata file format	474
Task metadata endpoint	476
Task metadata endpoint version 4	477
Task Metadata Endpoint version 3	493
Task Metadata Endpoint version 2	498
Container agent introspection	503
HTTP proxy configuration	504
Amazon Linux container instance configuration	505
Windows container instance configuration	507
Scheduling tasks	509
Run a standalone task	510
Task placement	513
Task groups	514
Task placement strategies	515
Task placement constraints	516
Cluster query language	520

Scheduled tasks	524
Create a scheduled task	524
View your scheduled tasks	526
Edit a scheduled task	526
Task lifecycle	527
Lifecycle states	528
Creating a scheduled task using the AWS CLI	529
Services	531
Service scheduler concepts	531
Daemon	532
Replica	533
Additional service concepts	533
Service definition parameters	534
Launch type	534
Capacity provider strategy	534
Task definition	535
Platform operating system	535
Platform version	536
Cluster	536
Service name	536
Scheduling strategy	537
Desired count	537
Deployment configuration	537
Deployment controller	539
Task placement	539
Tags	540
Network configuration	541
Client token	545
Service definition template	545
Creating a service	546
Creating a service using the new console	547
Creating a service using the classic console	548
Updating a service	559
Updating a service using the new console	560
Updating a service using the old console	561
Deleting a service	562
Deployment types	563
Rolling update	564
Blue/Green deployment with CodeDeploy	565
External deployment	569
Service load balancing	574
Service load balancing considerations	575
Load balancer types	576
Creating a load balancer	579
Registering multiple target groups with a service	589
Service auto scaling	591
Service auto scaling and deployments	591
IAM permissions required for service auto scaling	592
Target tracking scaling policies	593
Step scaling policies	597
Service Discovery	599
Service Discovery concepts	600
Service discovery considerations	601
Amazon ECS console experience	602
Service discovery pricing	602
Service throttle logic	602
Resources and tags	604

Tagging your resources	604
Tag basics	604
Tagging your resources	605
Tag restrictions	606
Tagging your resources for billing	606
Working with tags using the console	607
Working with tags using the CLI or API	609
Service quotas	611
Amazon ECS service quotas	611
AWS Fargate service quotas	613
Managing your Amazon ECS and AWS Fargate service quotas in the AWS Management Console ..	613
AWS Fargate Regions	614
Supported Regions for Linux containers on AWS Fargate	614
Supported Regions for Windows containers on AWS Fargate	615
Usage Reports	617
Monitoring	618
Monitoring tools	619
Automated Tools	619
Manual Tools	619
CloudWatch metrics	620
Enabling CloudWatch metrics	620
Available metrics and dimensions	621
Cluster reservation	623
Cluster utilization	624
Service utilization	625
Service RUNNING task count	626
Viewing Amazon ECS metrics	626
Tutorial: Scaling with CloudWatch Alarms	628
Events and EventBridge	632
Amazon ECS events	633
Handling events	644
CloudWatch Container Insights	645
Container Insights considerations	646
Setting up CloudWatch Container Insights for cluster and service level metrics	646
Container instance health	647
Collecting application trace data	648
Required IAM permissions	648
Specifying the AWS Distro for OpenTelemetry sidecar in your task definition	649
Logging Amazon ECS API calls with AWS CloudTrail	650
Amazon ECS information in CloudTrail	650
Understanding Amazon ECS log file entries	651
Security	653
Identity and access management	653
Audience	654
Authenticating With Identities	654
Managing access using policies	656
How Amazon Elastic Container Service works with IAM	658
Identity-based policy examples	663
AWS managed policies for Amazon ECS	674
Service-linked role	685
Task execution IAM role	691
Container instance IAM role	695
ECS Anywhere IAM role	697
IAM Roles for Tasks	699
CodeDeploy IAM Role	704
CloudWatch Events IAM Role	708
Troubleshooting	710

Logging and Monitoring	712
Compliance Validation	713
Infrastructure Security	714
Interface VPC endpoints (AWS PrivateLink)	714
Common use cases	718
Microservices	718
Auto Scaling	718
Service discovery	719
Authorization and secrets management	719
Logging	719
Continuous integration and continuous deployment	719
Batch jobs	720
Working with other services	721
Using Amazon ECR with Amazon ECS	721
Using Amazon ECR Images with Amazon ECS	721
Creating Amazon ECS resources with AWS CloudFormation	722
Amazon ECS and AWS CloudFormation templates	722
Learn more about AWS CloudFormation	722
Amazon Elastic Container Service on AWS Outposts	722
Prerequisites	723
Limitations	723
Network Connectivity Considerations	723
Creating an Amazon ECS Cluster on an AWS Outposts	723
Use App Mesh with Amazon ECS	726
AWS Deep Learning Containers on Amazon ECS	726
Deep Learning Containers with Elastic Inference on Amazon ECS	726
Tutorials	727
Tutorial: Creating a VPC	727
Step 1: Create an Elastic IP Address for Your NAT Gateway	727
Step 2: Run the VPC Wizard	728
Step 3: Create Additional Subnets	728
Next Steps	729
Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI	729
Prerequisites	729
Step 1: Create a Cluster	730
Step 2: Register a Linux Task Definition	730
Step 3: List Task Definitions	731
Step 4: Create a Service	732
Step 5: List Services	732
Step 6: Describe the Running Service	732
Step 7: Clean Up	734
Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI	734
Prerequisites	735
Step 1: Create a Cluster	735
Step 2: Register a Windows Task Definition	735
Step 3: List task definitions	737
Step 4: Create a service	737
Step 5: List services	737
Step 6: Describe the Running Service	738
Step 7: Clean Up	739
Tutorial: Creating a cluster with an EC2 task using the AWS CLI	739
Prerequisites	740
Step 1: Create a Cluster	740
Step 2: Launch an Instance with the Amazon ECS AMI	741
Step 3: List Container Instances	741
Step 4: Describe your Container Instance	741
Step 5: Register a Task Definition	743

Step 6: List Task Definitions	744
Step 7: Run a Task	745
Step 8: List Tasks	745
Step 9: Describe the Running Task	746
Tutorial: Using cluster auto scaling with the AWS Management Console	746
Prerequisites	747
Step 1: Create an Amazon ECS cluster	747
Step 2: Create the Auto Scaling resources	747
Step 3: Create a capacity provider	749
Step 4: Set a default capacity provider strategy for the cluster	761
Step 5: Register a task definition	761
Step 6: Run a task	762
Step 7: Verify	762
Step 8: Clean up	763
Tutorial: Specifying sensitive data using Secrets Manager secrets	764
Prerequisites	764
Step 1: Create an Secrets Manager secret	764
Step 2: Update your task execution IAM role	765
Step 3: Create an Amazon ECS task definition	766
Step 4: Create an Amazon ECS cluster	767
Step 5: Run an Amazon ECS task	767
Step 6: Verify	768
Step 7: Clean up	769
Tutorial: Creating a service using Service Discovery	769
Prerequisites	769
Step 1: Create the Service Discovery resources	769
Step 2: Create the Amazon ECS resources	771
Step 3: Verify Service Discovery	774
Step 4: Clean up	776
Tutorial: Creating a service using a blue/green deployment	778
Prerequisites	778
Step 1: Create an Application Load Balancer	778
Step 2: Create an Amazon ECS cluster	779
Step 3: Register a task definition	780
Step 4: Create an Amazon ECS service	780
Step 5: Create the AWS CodeDeploy resources	781
Step 6: Create and monitor a CodeDeploy deployment	783
Step 7: Clean up	785
Tutorial: Listening for Amazon ECS CloudWatch Events	787
Prerequisite: Set up a test cluster	787
Step 1: Create the Lambda function	787
Step 2: Register an event rule	787
Step 3: Test your rule	788
Tutorial: Sending Amazon Simple Notification Service alerts for task stopped events	788
Prerequisite: Set up a test cluster	788
Step 1: Create and subscribe to an Amazon SNS topic	788
Step 2: Register an event rule	789
Step 3: Test your rule	789
Tutorial: Using Amazon EFS	790
Step 1: Create an Amazon ECS cluster	791
Step 2: Create a security group for the Amazon EFS file system	792
Step 3: Create an Amazon EFS file system	792
Step 4: Add content to the Amazon EFS file system	793
Step 5: Create a task definition	794
Step 6: Run a task and view the results	795
Tutorial: Using FSx for Windows File Server	796
Prerequisites for the tutorial	796

Step 1: Create IAM access roles	797
Step 2: Create Windows Active Directory (AD)	797
Step 3: Verify and update your security group	798
Step 4: Create an FSx for Windows File Server file system	799
Step 5: Create an Amazon ECS cluster	799
Step 6: Create an Amazon ECS instance	800
Step 7: Register a Windows task definition	43
Step 8: Run a task and view the results	803
Step 9: Clean up	803
Troubleshooting	805
Using Amazon ECS Exec for debugging	805
Architecture	805
Considerations for using ECS Exec	806
Prerequisites for using ECS Exec	806
Enabling and using ECS Exec	807
Logging and Auditing using ECS Exec	809
Using IAM policies to limit access to ECS Exec	811
Troubleshooting issues with ECS Exec	814
Troubleshooting ECS Anywhere issues	814
External instance registration issues	814
External instance network issues	815
Issues running tasks	815
Checking stopped tasks for errors	815
CannotPullContainer task errors	817
Service event messages	819
Service event messages	821
Invalid CPU or memory value specified	824
CannotCreateContainerError: API error (500): devmapper	825
Troubleshooting service load balancers	826
Troubleshooting service auto scaling	827
Enabling Docker debug output	827
Amazon ECS Log File Locations	828
Amazon ECS Container Agent Log	829
Amazon ECS ecs-init Log	831
IAM Roles for Tasks Credential Audit Log	831
Amazon ECS logs collector	832
Agent introspection diagnostics	833
Docker diagnostics	835
List Docker containers	835
View Docker Logs	836
Inspect Docker Containers	836
AWS Fargate throttling limits	837
API failure reasons	837
Troubleshooting IAM Roles for Tasks	839
Amazon EC2 Windows containers	842
Windows container caveats	842
Getting started with Windows containers	843
Windows task definitions	843
Windows task definition parameters	843
Windows sample task definitions	845
Windows IAM roles for tasks	846
IAM roles for task container bootstrap script	847
Pushing Windows images to Amazon ECR	847
Building your own AMI	848
Listing the <code>ecs-optimized-ami-windows</code> component versions	849
Using gMSAs for Windows Containers	849
Considerations	850

Prerequisites	850
Setting Up gMSA-capable Windows Containers on Amazon ECS	850
Document history	854
AWS glossary	876

What is Amazon Elastic Container Service?

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster. Your containers are defined in a task definition that you use to run individual tasks or tasks within a service. In this context, a service is a configuration that enables you to run and maintain a specified number of tasks simultaneously in a cluster. You can run your tasks and services on a serverless infrastructure that is managed by AWS Fargate. Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon EC2 instances that you manage.

Amazon ECS enables you to launch and stop your container-based applications by using simple API calls. You can also retrieve the state of your cluster from a centralized service and have access to many familiar Amazon EC2 features.

You can schedule the placement of your containers across your cluster based on your resource needs, isolation policies, and availability requirements. With Amazon ECS, you don't have to operate your own cluster management and configuration management systems or worry about scaling your management infrastructure.

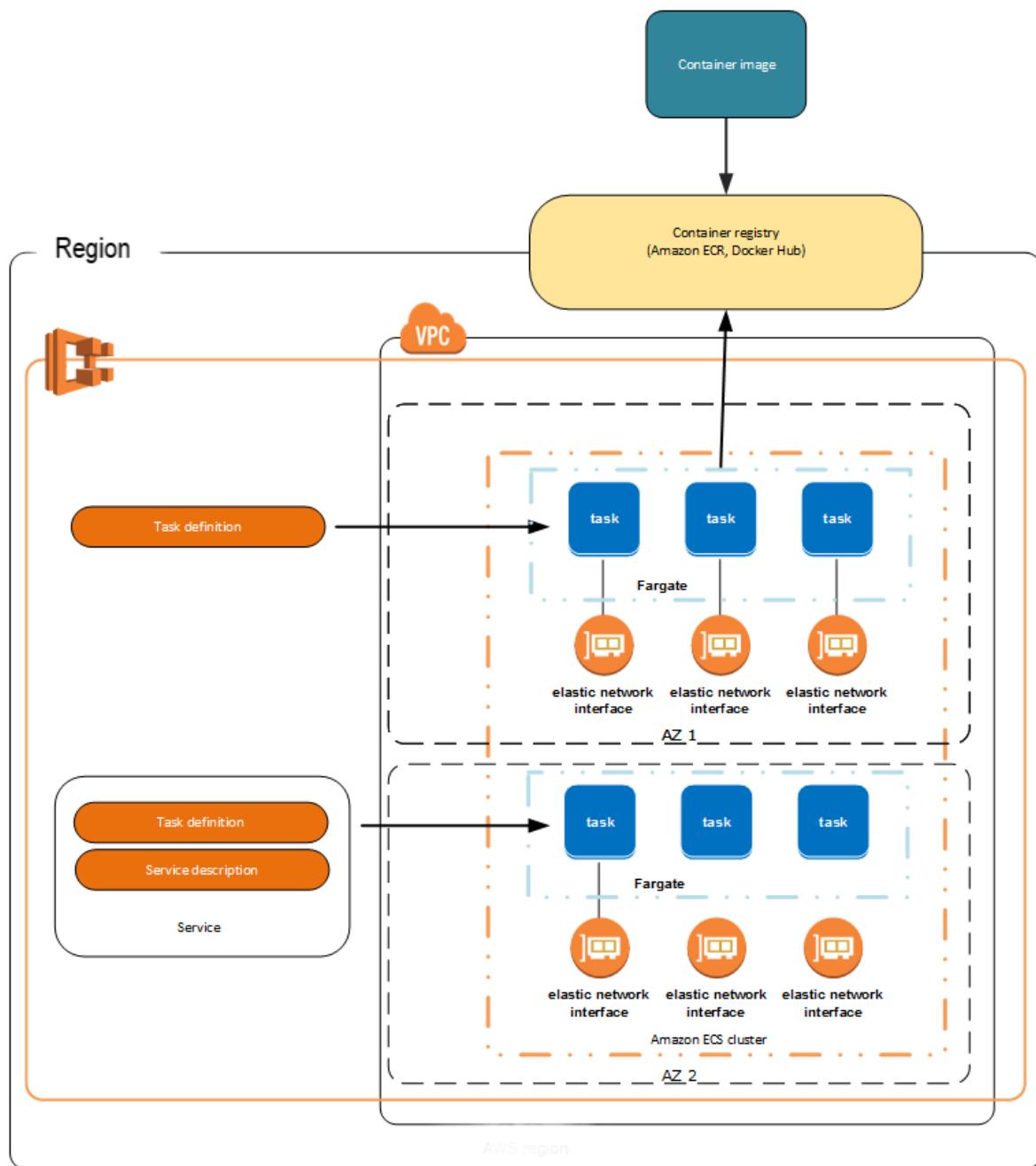
Amazon ECS can be used to create a consistent build and deployment experience, to manage and scale batch and Extract-Transform-Load (ETL) workloads, and to build sophisticated application architectures on a microservices model. For more information about Amazon ECS use cases and scenarios, see [Container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

The AWS container services team maintains a public roadmap on GitHub. The roadmap contains information about what the teams are working on and enables AWS customers to provide direct feedback. For more information, see [AWS Containers Roadmap](#).

Features of Amazon ECS

Amazon ECS is a regional service that simplifies running containers in a highly available manner across multiple Availability Zones within a Region. You can create Amazon ECS clusters within a new or existing VPC. After a cluster is up and running, you can create task definitions that define which container images run across your clusters. Your task definitions are used to run tasks or create services. Container images are stored in and pulled from container registries, for example, the [Amazon Elastic Container Registry](#).

The following diagram shows the architecture of an Amazon ECS environment run on AWS Fargate.

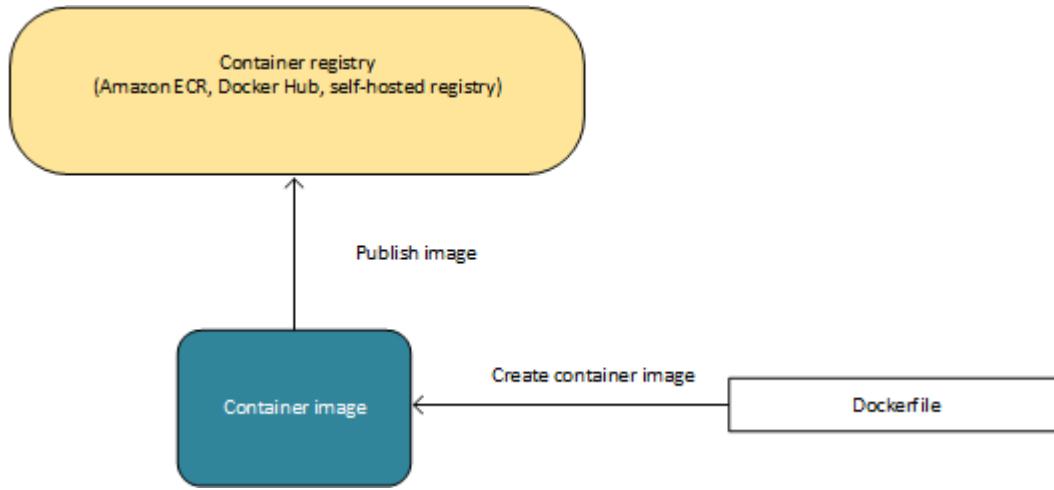


The following sections dive into these individual elements of the Amazon ECS architecture in more detail.

Containers and images

To deploy applications on Amazon ECS, your application components must be architected to run in *containers*. A container is a standardized unit of software development that contains everything that your software application needs to run, including relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template called an *image*.

Images are typically built from a Dockerfile, which is a plaintext file that specifies all of the components that are included in the container. After being built, these images are stored in a *registry* where they then can be downloaded and run on your cluster. For more information about container technology, see [Docker basics for Amazon ECS \(p. 14\)](#).



Task definitions

To prepare your application to run on Amazon ECS, you must create a *task definition*. The task definition is a text file (in JSON format) that describes one or more containers (up to a maximum of ten) that form your application. The task definition can be thought of as a blueprint for your application. It specifies various parameters for your application. For example, these parameters can be used to indicate which containers should be used, which ports should be opened for your application, and what data volumes should be used with the containers in the task. The specific parameters available for your task definition depend on the needs of your specific application. For more information about creating task definitions, see [Amazon ECS task definitions \(p. 194\)](#).

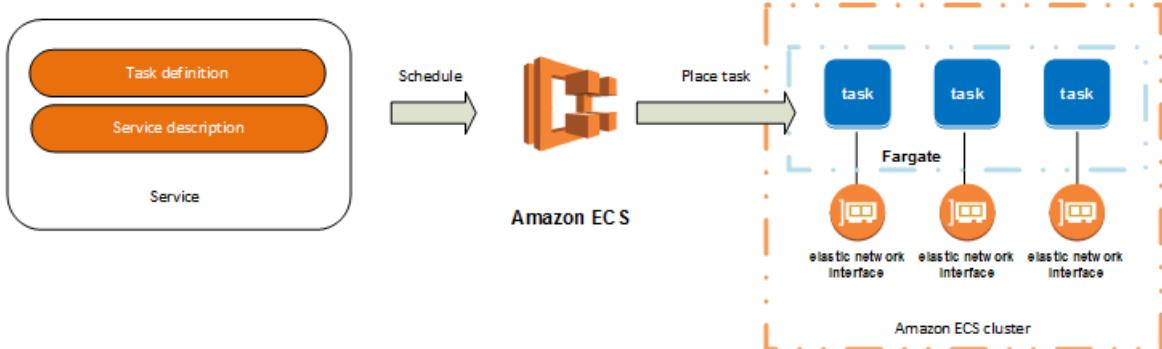
The following is an example of a task definition that specifies the use of Fargate to launch a single container that runs an NGINX web server. For a more extended example demonstrating the use of multiple containers in a task definition, see [Example task definitions \(p. 317\)](#).

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

Tasks and scheduling

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

The Amazon ECS task scheduler is responsible for placing tasks within your cluster. There are several different scheduling options available. For example, you can define a *service* that runs and maintains a specified number of tasks simultaneously. For more information about the different scheduling options available, see [Scheduling Amazon ECS tasks \(p. 509\)](#).



Clusters

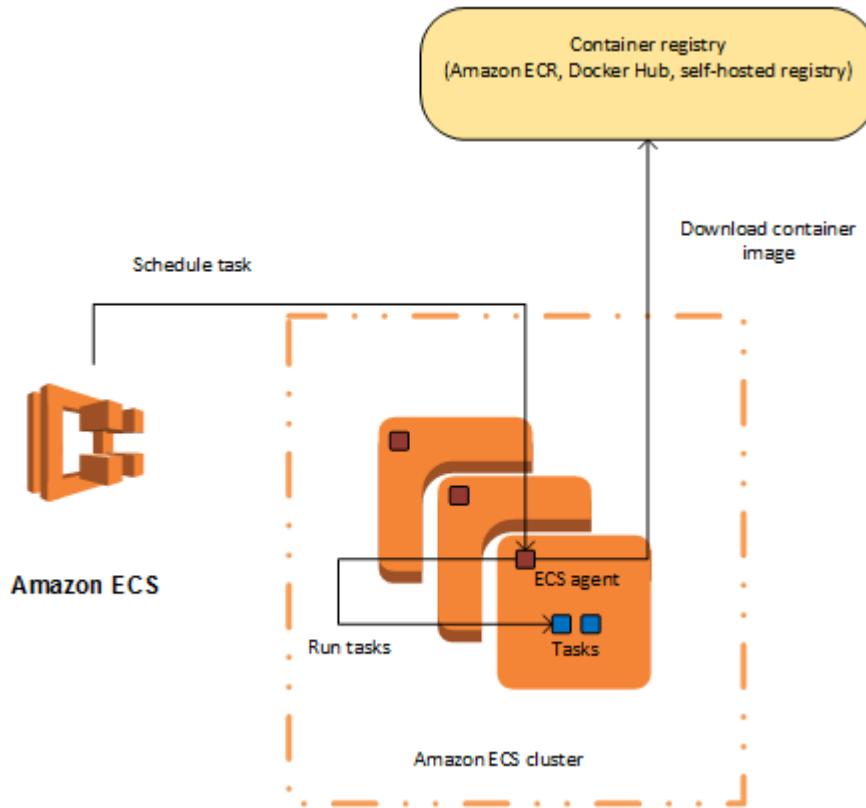
An Amazon ECS *cluster* is a logical grouping of tasks or services. You can register one or more Amazon EC2 instances (also referred to as *container instances*) with your cluster to run tasks on them. Or, you can use the serverless infrastructure that Fargate provides to run tasks. When your tasks are run on Fargate, your cluster resources are also managed by Fargate.

When you first use Amazon ECS, a default cluster is created for you. You can create additional clusters in an account to keep your resources separate.

For more information about creating clusters, see [Amazon ECS clusters \(p. 175\)](#). For more information about launching container instances and registering them with your clusters, see [Amazon ECS container instances \(p. 330\)](#).

Container agent

The *container agent* runs on each container instance within an Amazon ECS cluster. The agent sends information about the resource's current running tasks and resource utilization to Amazon ECS. It starts and stops tasks whenever it receives a request from Amazon ECS. For more information, see [Amazon ECS container agent \(p. 431\)](#).



Getting started with Amazon ECS

To learn about the developer tools available for using Amazon ECS, see [Amazon ECS developer tools overview \(p. 46\)](#).

If you are using Amazon ECS for the first time, the AWS Management Console for Amazon ECS provides a first-run wizard that steps you through defining a task definition for a web server, configuring a service, and launching your first Fargate task. We strongly recommend that you use the first-run wizard if you have little or no prior experience using Amazon ECS. For more information, see [Getting started with Amazon ECS \(p. 14\)](#).

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

Related services

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

AWS Identity and Access Management

IAM (Identity and Access Management) is an access management service that helps you securely control access to AWS resources. You can use IAM to control who is authenticated (signed in) and authorized (has permissions) to view or perform specific actions on resources. In Amazon ECS, you can use IAM to control access at the container instance level using IAM roles and at the task level using IAM task roles. For more information, see [Identity and access management for Amazon Elastic Container Service \(p. 653\)](#).

Amazon EC2 Auto Scaling

Auto Scaling is a service that enables you to automatically scale out or in your tasks based on user-defined policies, health status checks, and schedules. You can use Auto Scaling with a Fargate task within a service to scale in response to a number of metrics or with an EC2 task to scale the container instances within your cluster. For more information, see [Service auto scaling \(p. 591\)](#).

Elastic Load Balancing

The Elastic Load Balancing service automatically distributes incoming application traffic across the tasks in your Amazon ECS service. It enables you to achieve greater levels of fault tolerance in your applications, seamlessly providing the required amount of load-balancing capacity needed to distribute application traffic. You can use Elastic Load Balancing to create an endpoint that balances traffic across services in a cluster. For more information, see [Service load balancing \(p. 574\)](#).

Amazon Elastic Container Registry

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

AWS CloudFormation

AWS CloudFormation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources. More specifically, it makes resource provisioning and updating more orderly and predictable. You can define clusters, task definitions, and services as entities in an AWS CloudFormation script. For more information, see [AWS CloudFormation Template Reference](#).

Pricing

Amazon ECS pricing is dependent on whether you're using AWS Fargate or Amazon EC2 infrastructure to host your containerized workloads. When using Amazon ECS on AWS Outposts, the pricing follows the same model as when you're using Amazon EC2. For more information, see [Amazon ECS Pricing](#).

Amazon ECS and Fargate also offer Savings Plans that provide significant savings based on your AWS usage. For more information, see the [Savings Plans User Guide](#).

To view 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 additional 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](#).

Setting up with Amazon ECS

If you've already signed up for Amazon Web Services (AWS) and have been using Amazon Elastic Compute Cloud (Amazon EC2), you are close to being able to use Amazon ECS. The set-up process for the two services is similar. The following guide prepares you for launching your first 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 user**.
3. For **User name**, enter **Administrator**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **Administrators**.
10. Choose **Filter policies**, and then select **AWS managed - job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

Note

You must activate IAM user and role access to Billing before you can use the **AdministratorAccess** permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in [step 1 of the tutorial about delegating access to the billing console](#).

12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM entities](#) in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see [Access management](#) and [Example policies](#).

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

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

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

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. From the 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 key pair

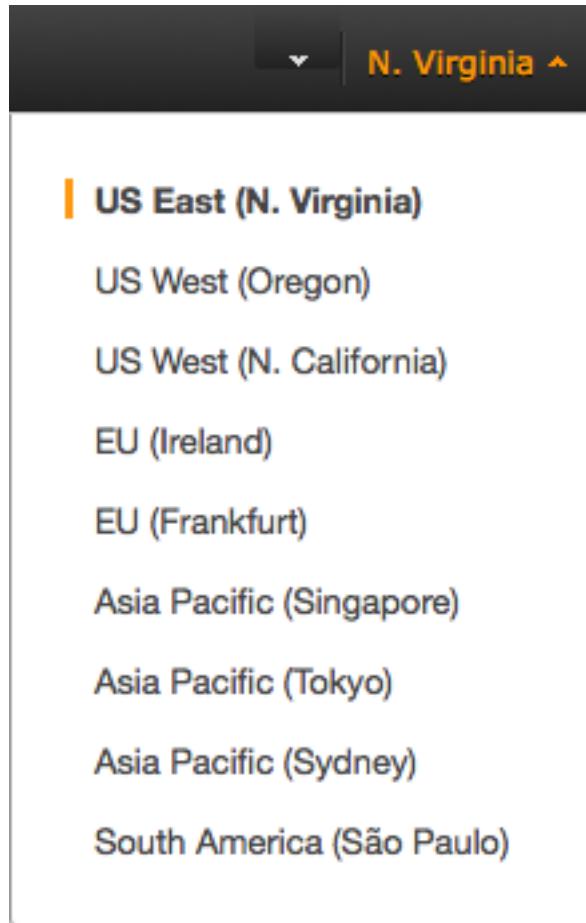
For Amazon ECS, a key pair is only needed if you intend on using the EC2 launch type.

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

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

To create a key pair

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select a Region for the key pair. You can select any Region that's available to you, regardless of your location. However, key pairs are specific to a Region. For example, if you plan to launch a container instance in the US East (Ohio) Region, you must create a key pair for the instance in the US East (Ohio) Region.



3. In the navigation pane, under **NETWORK & SECURITY**, choose **Key Pairs**.

Tip

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



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

Important

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

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

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

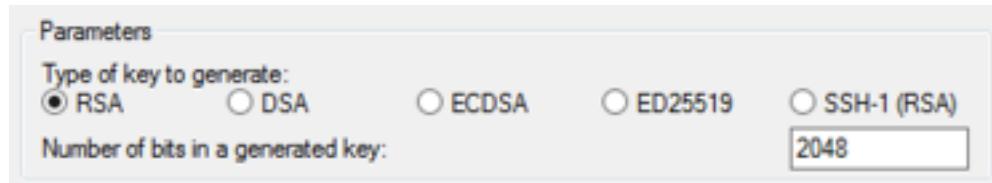
For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.

To connect to your instance using your key pair

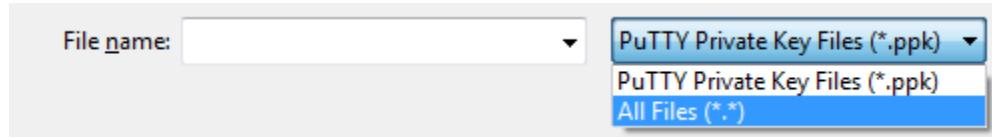
To connect to your Linux instance from a computer running macOS or Linux, specify the `.pem` file to your SSH client with the `-i` option and the path to your private key. To connect to your Linux instance from a computer running Windows, you can use either MindTerm or PuTTY. If you plan to use PuTTY,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

To prepare to connect to a Linux instance from Windows using PuTTY

1. Download and install PuTTY from <http://www.chiark.greenend.org.uk/~sgtatham/putty/>. Be sure to install the entire suite.
2. Start PuTTYgen (for example, from the **Start** menu, choose **All Programs > PuTTY > PuTTYgen**).
3. Under **Type of key to generate**, choose **RSA**.



4. Choose **Load**. By default, PuTTYgen displays only files with the extension **.ppk**. To locate your **.pem** file, select the option to display files of all types.



5. Select the private key file that you created in the previous procedure and choose **Open**. Choose **OK** to dismiss the confirmation dialog box.
6. Choose **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Choose **Yes**.
7. Specify the same name for the key that you used for the key pair. PuTTY automatically adds the **.ppk** file extension.

Create a virtual private cloud

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

Note

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

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

Important

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

To create a nondefault VPC

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

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

Create a security group

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

Note

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

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

Tip

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

To create a security group with least privilege

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select a Region for the security group. Security groups are specific to a Region, so you should select the same Region in which you created your key pair.
3. In the navigation pane, choose **Security Groups, Create Security Group**.
4. Enter a name for the new security group and a description. Choose a name that is easy for you to remember, such as *ecs-instances-default-cluster*.
5. In the **VPC** list, ensure that your default VPC is selected. It's marked with an asterisk (*).

Note

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

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

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

- Choose **HTTP** from the **Type** list, and make sure that **Source** is set to **Anywhere (0.0.0.0/0)**.
- Choose **HTTPS** from the **Type** list, and make sure that **Source** is set to **Anywhere (0.0.0.0/0)**.
- Choose **SSH** from the **Type** list. In the **Source** field, ensure that **Custom IP** is selected, and specify the public IP address of your computer or network in CIDR notation. To specify an individual IP address in CIDR notation, add the routing prefix /32. For example, if your IP address is 203.0.113.25, specify 203.0.113.25/32. If your company allocates addresses from a range, specify the entire range, such as 203.0.113.0/24.

Important

For security reasons, we don't recommend that you allow SSH access from all IP addresses (0.0.0.0/0) to your instance, except for testing purposes and only for a short time.

Install the AWS CLI

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

To use the AWS CLI with Amazon ECS, install the latest AWS CLI, version. For information about installing the AWS CLI or upgrading it to the latest version, see [Installing the AWS Command Line Interface](#) in the [AWS Command Line Interface User Guide](#).

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

- [Docker basics for Amazon ECS \(p. 14\)](#)
- [Getting started with Amazon ECS using AWS Copilot \(p. 18\)](#)
- [Getting started with Amazon ECS using the AWS CDK \(p. 23\)](#)
- [Getting started with the Amazon ECS console using Linux containers on AWS Fargate \(p. 30\)](#)
- [Getting started with the Amazon ECS console using Windows containers on AWS Fargate \(p. 33\)](#)
- [Getting started with the Amazon ECS console using Amazon EC2 \(p. 37\)](#)
- [Getting started with the Amazon ECS console using Amazon EC2 Windows containers \(p. 41\)](#)

Docker basics for Amazon ECS

Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications that are based on Linux containers. Amazon ECS uses Docker images in task definitions to launch containers as part of tasks in your clusters.

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](#).

Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see [Deploying Docker containers on Amazon ECS](#).

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

Installing Docker

Important

If you already have Docker installed, skip to [Create a Docker image \(p. 15\)](#).

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](#).

If you don't 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 and install Docker Engine and the Docker CLI.

To install Docker on an Amazon EC2 instance

1. Launch an instance with the Amazon Linux 2 or Amazon Linux 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.

Amazon Linux 2

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

Amazon Linux.

```
sudo yum install docker
```

5. Start the Docker service.

```
sudo service docker start
```

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

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

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

```
docker info
```

Note

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

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

Create a Docker image

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

To create a Docker image of a simple web application

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

```
touch Dockerfile
```

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

```
FROM ubuntu:18.04

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

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

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

EXPOSE 80

CMD /root/run_apache.sh
```

This Dockerfile uses the Ubuntu 18.04 image. The `RUN` instructions update the package caches, install some software packages for the web server, and then write the "Hello World!" content to the web server's document root. The `EXPOSE` instruction exposes port 80 on the container, and the `CMD` instruction starts the web server.

3. Build the Docker image from your Dockerfile.

Note

Some versions of Docker may require the full path to your Dockerfile in the following command, instead of the relative path shown below.

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

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

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

Output:

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

5. Run the newly built image. The `-p 80:80` option maps the exposed port 80 on the container to port 80 on the host system. For more information about `docker run`, go to the [Docker run reference](#).

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

Note

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

6. Open a browser and point to the server that is running Docker and hosting your container.
 - If you are using an EC2 instance, this is the **Public DNS** value for the server, which is the same address you use to connect to the instance with SSH. Make sure that the security group for your instance allows inbound traffic on port 80.
 - If you are running Docker locally, point your browser to <http://localhost/>.
 - If you are using **docker-machine** on a Windows or Mac computer, find the IP address of the VirtualBox VM that is hosting Docker with the **docker-machine ip** command, substituting **machine-name** with the name of the docker machine you are using.

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

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

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

Push your image to Amazon Elastic Container Registry

Amazon ECR is a managed AWS Docker registry service. 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](#).

This section requires the following:

- You have the AWS CLI installed and configured. If you do not have the AWS CLI installed on your system, see [Installing the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.
- Your user has the required IAM permissions to access the Amazon ECR service. For more information, see [Amazon ECR managed policies](#).

To tag your image and push it to Amazon ECR

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

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

Output:

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

```
    }
```

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

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

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

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

Output:

```
Login Succeeded
```

Important

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

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

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

Clean up

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

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

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 [Setting up with Amazon ECS \(p. 7\)](#).
- Install the AWS Copilot CLI. Releases currently support Linux and macOS systems. For more information, see [Installing the AWS Copilot CLI \(p. 50\)](#).
- 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.

Amazon Linux containers on Fargate

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

Windows containers on Fargate

```
git clone https://github.com/aws-samples/amazon-ecs-cli-sample-windows-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 tutorial 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 tutorial 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 tutorial, 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
```

Topics

- [Step 1: Set up your AWS CDK project \(p. 23\)](#)
- [Step 2: Use the AWS CDK to define a containerized Web server on Fargate \(p. 25\)](#)
- [Step 3: Test the Web server \(p. 29\)](#)
- [Step 4: Clean up \(p. 29\)](#)
- [Next steps \(p. 29\)](#)

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 this tutorial.

Now install the `aws-ecs-patterns` construct library module.

TypeScript

```
npm install @aws-cdk/aws-ecs-patterns
```

JavaScript

```
npm install @aws-cdk/aws-ecs-patterns
```

Python

```
python -m pip install aws-cdk.aws-ecs-patterns
```

Java

Edit the project's `pom.xml` to add the following dependencies inside the existing `<dependencies>` container.

```
<dependency>
  <groupId>software.amazon.awscdk</groupId>
  <artifactId>ecs</artifactId>
  <version>${cdk.version}</version>
</dependency>
```

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>ecs-patterns</artifactId>
    <version>${cdk.version}</version>
</dependency>
```

Maven automatically installs these dependencies the next time you build your app. To build, issue `mvn compile` or use your Java IDE's **Build** command.

C#

```
dotnet add src\HelloEcs package Amazon.CDK.AWS.ECS.Patterns
```

You may also install the indicated packages using the Visual Studio NuGet GUI, available via **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution**.

When you install the `aws-ecs-patterns` module, the `aws-ecs` module is also installed because it is a dependency of `aws-ecs-patterns`. You can use these modules in your AWS CDK app by importing the corresponding package.

TypeScript

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

JavaScript

```
@aws-cdk/aws-ecs
@aws-cdk/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/core';

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

export class HelloEcsStack extends cdk.Stack {
    constructor(scope: cdk.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/core');

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

class HelloEcsStack extends cdk.Stack {
    constructor(scope: cdk.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.

```
from aws_cdk import core as cdk

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

class HelloEcsStack(cdk.Stack):
```

```

def __init__(self, scope: cdk.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.amazon.awscdk.core.Construct;
import software.amazon.awscdk.core.Stack;
import software.amazon.awscdk.core.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 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.MyWebServerLoadBalancerDNSXXXXXX = Hello-MyWeb-ZZZZZZZZZZZZ-ZZZZZZZZZZ.us-
west-2.elb.amazonaws.com
HelloEcsStack.MyWebServerServiceURLYYYYYYYYY = http://Hello-MyWeb-ZZZZZZZZZZZZ-
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.



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 AWS 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 tutorial, see the AWS CDK API Reference overviews below.

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

Getting started with the Amazon ECS 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 Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks. In the Regions where Amazon ECS supports AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using the Fargate launch type. The wizard gives you the option of creating a cluster and launching a sample web application. If you already have a Docker image to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

Important

For information about getting started with Amazon ECS using Amazon EC2, see [the section called "Getting started with the Amazon ECS console using Amazon EC2" \(p. 37\)](#).

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

Prerequisites

Before you begin, be sure that you've completed the steps in [Setting up with Amazon ECS \(p. 7\)](#) and that your AWS user has either the permissions specified in the [AdministratorAccess](#) or [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.

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

- Your user has administrator access. For more information, see [Setting up with Amazon ECS \(p. 7\)](#).
- Your user has the IAM permissions to create a service role. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#).
- A user with administrator access has manually created the task execution role so that it is available on the account to be used. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

Step 1: Create a task definition

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

1. Open the Amazon ECS console first-run wizard at <https://console.aws.amazon.com/ecs/home#/firstRun>.
2. From the navigation bar, select the **US East (N. Virginia)** Region.

Note

You can complete this first-run wizard using these steps for any Region that supports Amazon ECS using Fargate. For more information, see [Amazon ECS on AWS Fargate \(p. 161\)](#).

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 \(p. 213\)](#).

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 \(p. 209\)](#).

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 \(p. 574\)](#).
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 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 Amazon Elastic Container Service? \(p. 1\)](#).

Get started with Amazon ECS on AWS Fargate by using the Fargate launch type for your tasks. In the Regions where Amazon ECS supports AWS Fargate, the Amazon ECS first-run wizard guides you through the process of getting started with Amazon ECS using the Fargate launch type. The wizard gives you the option of creating a cluster and launching a sample web application. If you already have a Docker image to launch in Amazon ECS, you can create a task definition with that image and use that for your cluster instead.

Prerequisites

Before you begin, be sure that you've completed the steps in [Setting up with Amazon ECS \(p. 7\)](#) and that your AWS user has either the permissions specified in the [AdministratorAccess](#) or [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.

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

- Your user has administrator access. For more information, see [Setting up with Amazon ECS \(p. 7\)](#).
- Your user has the IAM permissions to create a service role. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#).
- A user with administrator access has manually created the task execution role so that it is available on the account to be used. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

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 **CreateVPC**.
 - 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 \(p. 645\)](#).
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 `r`n'<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:** REPLICA
 - **Number of tasks:** 1
 - **Deployment type:** Rolling update

To create a service from your task definition with the AWS CLI

- Using the AWS CLI, run the following command to create your service.

```
aws ecs create-service --cluster windows --task-definition windows-simple-iis --  
desired-count 1 --service-name windows_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.



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.

Getting started with the Amazon ECS console using Amazon EC2

Amazon Elastic Container Service (Amazon ECS) is a fast and highly scalable container management service that makes it easy to launch and manage your containers. For a broad overview on Amazon ECS, see [What is Amazon Elastic Container Service? \(p. 1\)](#).

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

Important

For information about getting started with Amazon ECS using AWS Fargate, see [Getting started with the Amazon ECS console using Linux containers on AWS Fargate \(p. 30\)](#) or [Getting started with the Amazon ECS console using Windows containers on AWS Fargate \(p. 33\)](#).

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

Prerequisites

Before you begin, be sure that you've completed the steps in [Setting up with Amazon ECS \(p. 7\)](#) and that your AWS user has either the permissions specified in the [AdministratorAccess](#) or the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.

Step 1: Register a task definition

A task definition is like a blueprint for your application. Each time that you launch a task in Amazon ECS, you specify a task definition. The service then knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container. For more information about task definitions, see [Amazon ECS task definitions \(p. 194\)](#).

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

To register a task definition

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

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

```
        "name": "simple-app"
    },
],
"family": "console-sample-app-static"
}
```

7. Choose **Create**.

Step 2: Create a cluster

An Amazon ECS cluster is a logical grouping of tasks, services, and container instances. When creating a cluster using the console, Amazon ECS creates a AWS CloudFormation stack that takes care of the Amazon EC2 instance creation, networking and IAM configuration for you. For more information about clusters, see [Amazon ECS clusters \(p. 175\)](#).

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

To create a cluster

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. On the **Select cluster template** page, choose **EC2 Linux + Networking**.
6. For **Cluster name**, choose a name for your cluster.
7. In the **Instance configuration** section, do the following:
 - a. For **EC2 instance type**, choose either the **t2.micro** or **t3.micro** instance type to use for the container instance. Instance types with more CPU and memory resources can handle more tasks, but that is unnecessary for this getting started guide. For more information about the different instance types, see [Amazon EC2 Instances](#).
 - b. For **Number of instances**, type **1**. Amazon EC2 instances incur costs while they exist in your AWS resources. For more information, see [Amazon EC2 Pricing](#).
 - c. For **EC2 Ami Id**, use the default value which is the Amazon Linux 2 Amazon ECS-optimized AMI. For more information about the Amazon ECS-optimized AMI, see [Amazon ECS-optimized AMI \(p. 333\)](#).
8. In the **Networking** section, for **VPC** choose either **Create a new VPC** to have Amazon ECS create a new VPC for the cluster to use, or choose an existing VPC to use. For more information on creating your own VPC, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters \(p. 727\)](#).
9. In the **Container instance IAM role** section, choose **Create new role** to have Amazon ECS create a new IAM role for your container instances, or choose an existing Amazon ECS container instance (`ecsInstanceRole`) role that you have already created. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
10. Choose **Create**.

Step 3: Create a Service

An Amazon ECS service enables you to run and maintain a specified number of instances of a task definition simultaneously in an Amazon ECS cluster. If any of your tasks should fail or stop for any

reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it in order to maintain the desired number of tasks in the service. For more information on services, see [Amazon ECS services \(p. 531\)](#).

To create a service

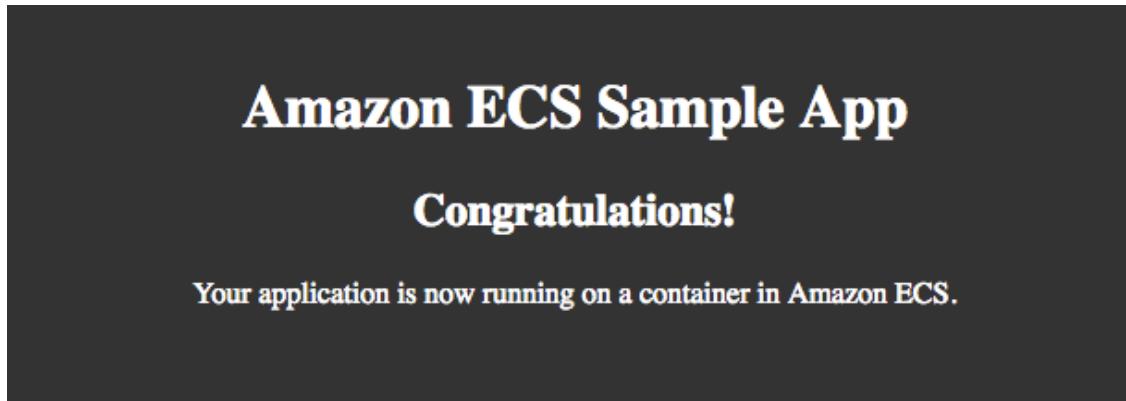
1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. Select the cluster you created in the previous step.
5. On the **Services** tab, choose **Create**.
6. In the **Configure service** section, do the following:
 - a. For **Launch type**, select **EC2**
 - b. For **Task definition**, select the **console-sample-app-static** task definition you created in step 1.
 - c. For **Cluster**, select the cluster you created in step 2.
 - d. For **Service name**, select a name for your service.
 - e. For **Number of tasks**, enter 1.
7. Use the default values for the rest of the fields and choose **Next step**.
8. In the **Configure network** section, leave the default values and choose **Next step**.
9. In the **Set Auto Scaling** section, leave the default value and choose **Next step**.
10. Review the options and choose **Create service**.
11. Choose **View service** to review your service.

Step 4: View your Service

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

To view the service details

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the same Region you used in the previous step.
3. In the navigation pane, choose **Clusters**.
4. Select the cluster you created step 2.
5. On the **Services** tab, choose the service you created in step 3.
6. On the **Service: *service-name*** page, choose the **Tasks** tab.
7. Confirm that the task is in a **RUNNING** state. If it is, select the task to view the task details. If it is not in a **RUNNING** status, refresh the service details screen until it is.
8. In the **Containers** section, expand the container details. In the **Network bindings** section, for **External Link** you will see the **IPv4 Public IP** address to use to access the web application.
9. Enter the **IPv4 Public IP** address in your web browser and you should see a webpage that displays the **Amazon ECS sample** application.



Step 5: Clean Up

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

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

To clean up the resources

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

Getting started with the Amazon ECS console using Amazon EC2 Windows containers

This tutorial walks you through getting Windows containers running on either Amazon ECS with the Amazon ECS-optimized Windows Server AMI in the AWS Management Console.

You create a cluster for your Windows container instances, launch one or more container instances into your cluster, register a task definition that uses a Windows container image, create a service that uses that task definition, and then view the sample webpage that the container runs. For more information, see [Amazon EC2 Windows containers \(p. 842\)](#).

Topics

- [Step 1: Create a Windows cluster \(p. 42\)](#)
- [Step 2: Register a Windows task definition \(p. 43\)](#)
- [Step 3: Create a service with your task definition \(p. 44\)](#)
- [Step 4: View your service \(p. 45\)](#)

Step 1: Create a Windows cluster

You can create a new cluster for your Windows containers. Amazon EC2 instances using the Linux Amazon ECS-optimized AMIs cannot run Windows containers, and vice versa, so proper task placement is best accomplished by running Windows and Linux container instances in separate clusters. In this tutorial, you create a cluster called windows and register one or more Amazon EC2 instances into the cluster for your Windows containers.

To create a cluster with the AWS Management Console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose **Create Cluster**.
4. Choose **EC2 Windows + Networking** and choose **Next step**.
5. For **Cluster name** enter a name for your cluster (in this example, windows is the name of the cluster). Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. In the **Instance configuration** section, complete the following steps.
 - a. For **Provisioning model**, choose one of the following instance types:
 - **On-Demand Instance**– With On-Demand Instances, you pay for compute capacity by the hour with no long-term commitments or upfront payments.
 - **Spot**– Spot Instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. For more information, see [Spot Instances](#).

Note

Spot Instances are subject to possible interruptions. We recommend that you avoid Spot Instances for applications that can't be interrupted. For more information, see [Spot Instance Interruptions](#).

- b. For Spot Instances, do the following; otherwise, skip to the next step.
 - i. For **Spot Instance allocation strategy**, choose the strategy that meets your needs. For more information, see [Spot Fleet Allocation Strategy](#).
 - ii. For **Maximum bid price (per instance/hour)**, specify a bid price. If your bid price is lower than the Spot price for the instance types that you selected, your Spot Instances are not launched.
 - c. For **EC2 instance type** page, choose the hardware configuration of your instance. The instance type that you select determines the resources available for your tasks to run on.
 - d. For **Number of instances**, choose the number of Amazon EC2 instances to launch into your cluster.
 - e. For **EC2 AMI Id**, choose the Amazon ECS-optimized AMI to use for your container instances. The available AMIs will be determined by the Region and instance type you chose. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).
 - f. For **EBS storage (GiB)**, choose the size of the Amazon EBS volume to use for data storage on your container instances. You can increase the size of the data volume to allow for greater image and container storage.
 - g. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for RDP access. If you do not specify a key pair, you cannot connect to your container instances with RDP. For more information, see [Amazon EC2 Key Pairs](#) in the [Amazon EC2 User Guide for Linux Instances](#).
7. In the **Networking** section, configure the VPC to launch your container instances into. By default, the cluster creation wizard creates a new VPC with two subnets in different Availability Zones, and

a security group open to the internet on port 80. This is a basic setup that works well for an HTTP service. However, you can modify these settings by following the substeps below.

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

Note

You can also choose to create a new security group and then modify the rules after the cluster is created. For more information, see [Amazon EC2 security groups for Windows instances](#) in the *Amazon EC2 User Guide for Windows Instances*.

- e. In the **Container instance IAM role** section, select the IAM role to use with your container instances. If your account has the **ecsInstanceRole** that is created for you in the console first-run wizard, it is selected by default. If you do not have this role in your account, you can choose to create the role, or you can choose another IAM role to use with your container instances.

Important

The IAM role you use must have the **AmazonEC2ContainerServiceforEC2Role** managed policy attached to it, otherwise you will receive an error during cluster creation. If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent does not connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

- f. If you chose the Spot Instance type earlier, the **Spot Fleet Role IAM role** section indicates that an IAM role **ecsSpotFleetRole** is created.
- g. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see [Tagging Your Amazon ECS Resources](#).
- h. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 645\)](#).
- i. Choose **Create**.

Note

It can take up to 15 minutes for your Windows container instances to register with your cluster.

Step 2: Register a Windows task definition

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

To register the sample task definition with the AWS Management Console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Task Definitions**.
3. On the **Task Definitions** page, choose **Create new Task Definition**.
4. On the **Select launch type compatibilities** page, choose **EC2**, **Next step**.
5. Scroll to the bottom of the page and choose **Configure via JSON**.

6. Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose **Save**.

```
{  
  "family": "windows-simple-iis",  
  "containerDefinitions": [  
    {  
      "name": "windows_sample_app",  
      "image": "mcr.microsoft.com/windows/servercore/iis",  
      "cpu": 512,  
      "entryPoint": ["powershell", "-Command"],  
      "command": ["New-Item -Path C:\\inetpub\\wwwroot\\index.html -ItemType file -Value '<html> <head> <title>Amazon ECS Sample App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon ECS.</p>' -Force ; C:\\ServiceMonitor.exe w3svc"],  
      "portMappings": [  
        {  
          "protocol": "tcp",  
          "containerPort": 80,  
          "hostPort": 8080  
        }  
      ],  
      "memory": 768,  
      "essential": true  
    }  
  ]  
}
```

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

To register the sample task definition with the AWS CLI

1. Create a file called `windows-simple-iis.json`.
2. Open the file with your favorite text editor and add the sample JSON above to the file and save it.
3. Using the AWS CLI, run the following command to register the task definition with Amazon ECS.

Note

Make sure that your AWS CLI is configured to use the same region that your Windows cluster exists in, or add the `--region` `your_cluster_region` option to your command.

```
aws ecs register-task-definition --cli-input-json file://windows-simple-iis.json
```

Step 3: Create a service with your task definition

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

To create a service from your task definition with the console

1. On the **Task Definition: windows-simple-iis** registration confirmation page, choose **Actions, Create Service**.
2. On the **Create Service** page, enter the following information and then choose **Create service**.
 - **Launch type:** EC2
 - **Cluster:** windows

- **Service name:** windows-simple-iis
- **Service type:** REPLICA
- **Number of tasks:** 1
- **Deployment type:** Rolling update

To create a service from your task definition with the AWS CLI

- Using the AWS CLI, run the following command to create your service.

```
aws ecs create-service --cluster windows --task-definition windows-simple-iis --  
desired-count 1 --service-name windows-simple-iis
```

Step 4: View your service

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

Note

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

To view your service

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, choose the **windows** cluster.
3. In the **Services** tab, choose the **windows-simple-iis** service.
4. On the **Service: windows-simple-iis** page, choose the task ID for the task in your service.
5. On the **Task** page, expand the **iis** container to view its information.
6. In the **Network bindings** of the container, you should see an **External Link** IP address and port combination link. Choose that link to open the IIS test page in your browser.

Amazon ECS Sample App

Congratulations!

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

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 \(p. 46\)](#).
- Perform specific actions to Amazon ECS resources with automated deployments through programming or scripts using the [AWS Command Line Interface \(p. 46\)](#), [AWS SDKs \(p. 49\)](#) or the ECS API.
- Define and manage all AWS resources in your environment with automated deployment using [AWS CloudFormation \(p. 47\)](#).
- Use the complete [AWS Copilot CLI \(p. 47\)](#) 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 \(p. 47\)](#).
- Containerize applications that are hosted on premises or on Amazon EC2 instances or both by using the [AWS App2Container \(p. 48\)](#) 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 \(p. 48\)](#).
- Launch containers from [Docker Desktop integration with Amazon ECS \(p. 48\)](#) 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 \(p. 14\)](#).

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 \(p. 722\)](#).

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 \(p. 49\)](#).

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 \(p. 23\)](#).

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 \(p. 54\)](#).

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 signup](#).

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

```
brew install gnupg
```
 - For Linux systems, install gpg using the package manager on your flavor of Linux.
 - For Windows systems, download and use the Windows simple installer from the GnuPG website. For more information, see [GnuPG Download](#).
2. Create a local file with the following contents of the Amazon ECS PGP public key and then import it.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: GnuPG v2  
  
mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU  
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRM02f174lmaavr4Vg  
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWfdKuxoPCttBqoMj3LGn6Pe+6xVWRkChQu  
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx  
N+6pHBjRkIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uychKeliCrw4kj0+skizBgO  
2K7oVX80c3j5+ZilhL/qDLXmuCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNiCo  
1FTTrT7QwD5fMN1d3FanLgv/ZnIrsSaqJ0L6zRSq804N10WBVBndExk2Kr+5kFxN  
51BPgfPgRj5h0+KTHMa9Y8Z7yUc64Bj1N6F9N17FJuSsfqbdkvRLsQRcbcBG9qxX3  
rJAEhieJzVMEUN1+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzBq  
1TnWWCz4xrIWtuEBA2qE+MldheVd78a3gIsEaSTfQdQosYXaQbvlnSWOoc1y/5Zb  
zizHTJIhLtUyIs9WisP2s0emeHZicVMfW61EgPrJAIupgc7kyZvFt4YwfwARAQAB  
tCRBbWF6b24gRUNTIDx1Y3Mtc2VjdXJpdH1AYW1hem9uLmNvbT6JAhwEEAECAAYF  
Alrjl0YACgkQHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7  
Bs3pcPp8BV7BdRbs3sPlt5bV1+rkq0lw+0gZ4Q/ue/YbWtOAt4qY00cEo0HgcnaX  
1sB827QifZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmaZ  
McWB4HUMNrh0JgBCo0gIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF  
40oISF6hmQKIBoLnRrAlj5Qvs3GhvHQThYq0Grk/KMJJX2CSqt7twJ8gk1n3H3Y  
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBuaoZy5hF6maD09nHcnvBjqADzeT8Tr/Qu  
bBCLzkNSYqqkpgtwv7seod2P4n1giRvdAOEfMZpVkrUr+C252IaH1HZFEz+TvBVQM  
Y8OWWxmIJW+J6evjo3N1e019Uhv71jvoF8z1jbI4bsL2c+QTJmOv7nRqzDQgCWyp  
Id/v2dUVVTk1j9omuLBbWnjZQCB+72LcIzJhYmaP1HC4LcKQG+/f41exuItenatK  
1EJQhYtyVXcBlh6Yn/wzNg2NW0wb3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz  
N2HqkTSQh77Z8KPkmGopsrn/reMu1PdINb249nA0dz0N+nj+tTFOYCiaLaFyjs  
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHcwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX  
gAAKCR86dmkLvf4T91FEACenkmlndNxswUx34R3c0vamHrPxvflyI1F1EUen8D1h  
ux9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAkg214QRVzft0y9/DdR+twApA0fzy  
uavIthGd6+03jAAo6udyDE+cZC3P7XBbDiYEWk4XAf9I1JjB8hTZUgvXBL046JhG
```

eM17+crgUyQeetkiOQemLbsbxQ40Bd9V7zf7XJraFd8VrwNUwNb+9KFtgAsc9rk+
YIT/PEf+YOPysgcxI4sTWghtyCulVnuGoskgDv4v73PALU0ieUrVvQVqWMRvhVx1
0X90J7cC1KOyhLEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOrlZ3+6xBIm/qe
bFyLUhf4WoiuOplaAaJhK9pRY+XEenGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34FlGr
KVHUq1TZD7cvMnnNKEELTUCXK+1mV3an16nmAg/my1JSut6BNK2rJpY1s/kkSGSE
XQ4zuF21GCPvBFhYAlt5Un5zwqkwwQR3/n2kwAoDzonJcehdw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqrT9X42yF1PIEVrNeBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+ekr5TpVsZS9few2GpI5bCgBKBisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrrARAxAxNPvVwreJ2yAiFcUpdR1VhsuOgnxvs1OgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMuCIINTH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+J1hZwd8Mx2K+LVVVu/aWknrfMuNwyDucISI4D5QHa8T+F8fgN4OTpwYjirzel
5yoICMr9hVcbzDNv/oZKCxjx+XKgnFc3wrnDfjfntDAT7ecwbUTL+viQKJ646s+
psiqXRytVvYInEhLvrJ0aV6zHFOigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9CAXfeSmf77I+XTifigNna8x
t/ModjXr1fjF4pThEi5u6WsuRdfwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
2iHiOKIpQqLbHEfQmHcd2fix+AajKmnpGNku9qCFEmbgSRJpXz6BfwN1QoKE+i
R6ja0frUNt2jhiGG/F8RceXzohaaC/Cx7LUCUFwC0n7z32C9/Dtj7I1PMoacdZzz
bjjZrKO/ZDv+UN/c9dwAk11zAyPMwGBkUaY68EBstnIliW34aWm6iHhxioVPKSp
VJfyiXP00EXqujtHLAeChfjcns3I12YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIAQUCWrVJqwIbDAAKCRC86dmkLvf4T+ZdD/9x/8APzgNJF3o3STR
jvnV1ycyhVQGAeBJiu7wjsNWWzMAF0v15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
X7DR0Jszah9wrYTzXLVruAu+t6UL0y/XQ4L1GZ9Q6+r+7t1Mvbfy7B1HbvX/gYt
Rwe/uwdib10CagEzyX+2D3kT01HO5XThbXaNF8AN8za91Jt2Q2UR2X5T6JcwtMz
FBvZn13LSmZyE0E0ehS2iUurU4uWOpGppuqVnbi0jbCvCHKgDGrgZ0smKNAQng54
F365W3g8AfY48s8XQwzmc1iowYX9bT8Pz1e0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDj15Ca9pGpJXrC3xe
TXikQ8DBWDhBPVPruruLlaenTtZEosPc4I85yt5U9RoPTStcOr34s3w5yEaJagt6S
Gc5r9ysjkH6+6rbilujxmGROSqtqr+RyB+V9A5/OgtNZc811K6u4UoOCde8jUUW
vqWKVkjJB/Kz3u4zaeNu2ZyyHa0qOuH+TETCw+jsY91hbEzqN5yQYGi4pVmDkY5vu
1XbJnbqPKpRXdgM9BecV9AMbPgbDq/5LnHJJXg+G8YQ0gp41R/hC1TEFdIp5wM8AK
CwsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUuSARAAr77kj7j2QR2SzeOS1FBvV7oS
mFeSNnz9xZssqrsm6bTwSHM6YLDwc7Sdf2esDdyzONETwqrVCg+Fxg18hmo9hs4c
rR6tmrP0m0mrptr+xLlsKcaP7ogIXsyZnrEAESw8PnfayoipCdc3cMCR/1tnHFga
7EuR/XLBmi7Qg9tByVQYQ5jwB9V4B2yeCt3XtzPqeLkvaxl7PNelaHGJQY/xo+m
V0bndxf91Y+4oFJ4b1D32WqvyxEso7vW6Bh7oqv3zbm0yQrr8a6mDbpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSSGm4U/G4JB+JIG1ADY+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTss0G+300cGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikAO1vE3/wgMqCXscbycbLjLD/bXiUFWo3rzoezeXjgi/DJx
jKBAyBTY05nMcth109oaFd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rXl07Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRxWdxJwno5awTwXKRJMxGfhCvSvbcbc2Wx+L
IKvmB7E84K3fmjFFE67yolminw2qRcUBfygtH3eL5XXU28MiCpue8Y8GKJoBAUyvf
KeM1r08Jm3iRac5a/DOAEQEAAYkEPgQYAQIAQUCWrVLkgIbAgIpCRC86dmkLvf4
T8FdiAQZAIAbgUCWrVlkgAKCRDePL1hra+LjtHYD/9MucxdFe6bX01dQR4tKhhQ
POLRqy6z1B9Y1LCLowNdGZdqorogUiUymgn3VhEtvxTOoHcN7qOuM01PNsRnOeS
EYjf8Xrb1clzkD6xULwmOcltb9bBxnbc/4PFvHAbZW3QzusaZniNgkuxt6BtfloS
Of4inq71kjmgK+T1zQ6nUMQug228NUQC+a84EPqYyAeY1sgvgB7hJbhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAbf0OvfHfRBp0+m0ZwnJM8cPRFjOqqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNv7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmsMTJOPGn5SB1HyjCZNx6bhIIbQyEUB9gKCMuFaqXKwKpF6rj0
iQXAjxLR/shZ5Rk96VxzOphU17T90m/PnueEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmrOqX9zBCVxh0mdWYlrWvmzQFWzG7Aoe55fkf8nAEpsalrCdtaNubhrxa00QxG
AHModJQ0vBsmqMvuAdjkDwpFu5y0My5ddu+hiUzUyQlJ5Hhd5L0UDdewlZgIw1j
xrEAUzDKeTnem8GkHxDgg8koev5frmShJuce7vSjkPcnG3EIJSggMOPFjJuLWTz
vjHeDnbJy6uNL65ckJy6WhgeADS2WA1D6Tfekkc21sSIK/LqEpLMR/0g50Uif
wcEN1rS91JXbwIy8Me1N9qr5KcKQlmdfBNEyyceBhyV10MDyHOKC+7PofMtkGBq
13QieRHv5GJ8LB3fc1qHv8pwTT03Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRyKtoTjwob73CGkBZZpJyqux/rmCV/fp4ALdSW8bz
FJVRaih0WwzjpFQKhwcu91ABxi2UvVm14v0Afei70iJPSU1zM4fEny4oiIBX1R
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMi+mWra23EwjChaxpvjjcUH
51Lc5zq781aCYRygYQw+hu5nFkOH1R+Z50Ubxjd/aqUfnGIAX7kPMD3Lof4K1dD
Q8ppQriUvxVo+4nPv6rpTy/PyqCLWDjkguHpsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCE17nbFrm0Viia75Sa8KnywTdsyZsu3XcOcf3g+g1xWtpjJqy2bYX1qz
9uDOWtArWHOis6bq819RE6xr1RBVxs6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMbda64e9Rq7bFR7Tvt8gasCZY1Nr3lydh+dFHIEkH53HzQe6188HEic
+0jVnLkCDQRa55wJARAAyLy2aLx6gyoWoJN1a6740q3o8e9d4KggQoFGMTcflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhBHerZdbNjyjMNT1gAgrhPNB4HtXBxum2wS57WK

```

DNmade914L7FWTPAWBG2Wn448OEHTqsC1ICXXW9IIcgclAEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xOIZQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ10wuUidgfPCSVoUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKf89G72xdv8ut9AYYQ2BbEYU+JAYhUh8rYYui2dHKJigjNvJscuUwb
+QEgJIRleJRhro+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjziRceuJCJYox1cwsq6jTE50GinNzcIxTn9xUc0UMKFeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm91hkGwgTZgOlRX5eqPX+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6SYKqroYwBGGZm6Js5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9zlw8qsA
EQEEAAyKEPgQYAQgACQUCWuecCQIBAgIpCRC86dmkLVF4T8FDIAQZAQgABgUCWuec
CQAKCRBQ3szEc05hr+ykD/4tOLRHFXuUCxgGaUbUcvtsFrwBKma1cYjqapms8u
6Sk0wfGRI32G/GhOrp0Ts/MOkbObq6VLTh8N5Yc/53ME18zQFw9Y5AmRoW4PZXER
ujss5s7p40R7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEFHYT8UTxwnGmXovNkMM7md
xD3CV5q6VAt8WKBo/220II3fcQlc9r/oWX4kXXlbo9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW/NN2gju2G3Lu/T1FUWIT4v/50PK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUhQWeSNHI+fpe5oX6FALPT9JLDce80zz1i
cZZ0MELP37mOOQun0AlmHm/hvzf0f311PtbcqWaE51tJvgUR/nZFo6Ta3O5Ezs
3V1EJNQ11jf/6DH87SxvAoRIARCuZd0qxBcDK0avpFzUtbJd241RA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LeVfk6E8oLpxiegS4fiqC16mFrOdyRk+RJfIUyz0WTDVm
g0U1CO1ezokMSqkj7724pyjr2xf/r9/sC6aOJwB/1KgZkJfC6NqL7TlxVA31dUga
LEOvEJTTE4gl+tYtFcsCDvALCtqL0jduSkUo+RXcBItmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNm9mw9ydi1yjYXX5a9x4wMjracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6M1I2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBXuAx7+HgPPSFtrHQONCALxxzlbNpS+zxt9rOMiLgcLyspWxSdmoYGZ6nQP
R05Nm/ZVS+u2imPCRzNUZEMa+d1E6kHx0rS0dPiuj407NtPeYDKkoQtNagspsDv
cK7CSqAiKMq06UBTxqlTSRkm62e0Ctcs3p30eHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7X+451cCfmCvt94TFNL5HwEUJVpm0gmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3m0QGmNJ3bCLuc/jq7ysGq69xiKmTlUeXFm+aojcRO5i
zyShIRJZ0GZfuzDYFDbMV9ama/YQGygLw//zP5ju5SW26dNx1f3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRG1U/LpNSefnvDFTtEIRcpOHc
bhayG0bk51Bd4mioOXnIsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAMj20+X+j
qX2yy/UX5nSPU492e2CdZ1Uhu0USRFY3bxKHKB7SDbVeav+K5g==

=Gi5D
-----END PGP PUBLIC KEY BLOCK-----

```

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

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

- Import the Amazon ECS PGP public key with the following command.

```
gpg --import <public_key_filename>
```

- 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
```

5. 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](#).

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 \(p. 18\)](#).

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 \(p. 49\)](#).

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 \(p. 18\)](#).

Topics

- [Installing the Amazon ECS CLI \(p. 55\)](#)
- [Configuring the Amazon ECS CLI \(p. 60\)](#)
- [Migrating Configuration Files \(p. 61\)](#)
- [Tutorial: Creating a cluster with a Fargate task using the Amazon ECS CLI \(p. 62\)](#)
- [Tutorial: Creating a Cluster with an EC2 Task Using the Amazon ECS CLI \(p. 67\)](#)
- [Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI \(p. 70\)](#)
- [Amazon ECS command line reference \(p. 73\)](#)

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 \(p. 49\)](#).

Follow these instructions to install the Amazon ECS CLI on your macOS, Linux, or Windows system.

Step 1: Download the Amazon ECS CLI

Download the Amazon ECS CLI binary.

- For macOS:

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

- For Linux systems:

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

- For Windows systems:

Open Windows PowerShell and run the following commands:

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

Note

If you encounter permissions issues, ensure that you are running PowerShell as an administrator.

Step 2: Verify the Amazon ECS CLI 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](#).

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

```
brew install gnupg
```

- For Linux systems, install gpg using the package manager on your flavor of Linux.
- For Windows systems, download and use the Windows simple installer from the GnuPG website. For more information, see [GnuPG Download](#).

2. Create a local file with the following contents of the Amazon ECS PGP public key and then import it.

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

mQINBFq1SasBEADliGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2f174lmavr4Vg
7K/KH8VH1q2uRw32/B94XLEgRbGTMdWfdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnL3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGzoTx
N+6pHBjrKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uychKeliCrw4kj0+skizBgO
2K7oVX80c3j5+ZilhL/qDLXmuCzbaz5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNICo
1FTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq804LN10WBVbndExk2Kr+5kFxn
51BPgfPgRj5hQ+KTHMa9Y8Z7yUc64Bj1N6F9N17FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUN1+EgeCxkj5xsuKnu7zw2c3hQzqEcrADLV+hvFJktOz9Gm6xzBq
1TnWWCz4xrIWTuEBA2qE+M1DheVd78a3gIsEaSTfQq0osYXaQbvlnSWOoc1y/5Zb
zizHTJlhLtUyls9Wis2s0emeHZicVMFW61EgPrJaiupgc7kyZvFt4YwfwARAQAB
tCRBbWF6b24gRUNTIDx13Mtc2VjdXJpdH1AYW1hem9uLmNvbT6JAhwEEAECAAyF
AlrjL0YAcgkOHivRXs0TaQrg1g/+JppwPqHnlVPmv7lessB8I5UqZeD6p6uVpHd7
Bs3pcPp8BV7BdRbs3sPlt5bV1+rkgQ0lw+0gZ4Q/ue/YbWtOAt4qY00cEo0HgcnaX
lsB827QifZIVtGWMhuh94xzm/SJkvngml6KB3YJNnWP61A9qJ37/VbVVLzvcmaZ
McWB4HUMNrhd0JgBCo0gIpqCbpJEvUc02Bjn23eEJs9kC7OUAHyQkVnx4d9UzXF
4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJJX2CSqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgT6r5Q2HW1TBuaoZy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtwv7seod2P4n1giRvDAOEfMZpVkUr+C252IaH1HZFEz+TvBVQM
Y8OWWxmIJW+J6evjo3N1eO19UhV71jvoF8z1jbI4bsL2c+QTJmOv7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBBwNJzQCB+72LcIzJhYmaP1HC4LcKQG+/f4lexuItentatK
1EJQhYtyVXcBlh6Yn/wzNg2NWObw3vqY/F7m6u9ixAwgtIMgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPkmYGopsM/reMu1PdInb249nA0dzoN+nj+tTFOYCiaLaFyjs
Z0r1QAOJAjkEEwECACMFAlq1SasCGwMHcwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIX
```

gAAKCRC86dmkLVF4T9iFEACEEnkm1dNXsWUx34R3c0vamHrPxvfkyI1F1EUen8D1h
ux9xy6jCEROHWEp0rjGK4QDPgM93sWJ+s1UAKg214QRVzft0y9/Ddr+twApA0fzy
uavIthGde+03jAAo6udyDE+cZC3P7XBbDiYEWk4XAF9I1JjB8hTZugvXBL046JhG
eM17+crgUyQeetki0QemLbsbXQ40Bd9V7zf7XJraFd8VrwNuNb+9KftgAsc9rk+
YIT/PEF+YOPysgcxi4stWghtyCulVnuGoskgDv4v73PALU0ieUrvvQVqWMRvhVx1
0X90J7cc1KOyh1EQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOr1z3+6xBIm/qe
bFyLUnf4WoiuOp1AaJhK9pRY+XEnGNxdtN4D26Kd0P+PLkm3Tr3Hy3b1Ok34F1Gr
KVHUq1TZD7cvMnnKEELTucKX+1mV3an16nmAg/my1JSUt6BNK2rJpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5un2wqkwQR3/n2kwAoDzonJcehdw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqr9X4yF1PIEVnreBRJ3HDezAgJrNh0GQWRQkhIx
gz6/cTR+eR5TptVsS9few2GpI5bCgBKisZIssT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrrARAxNPvWreJ2yAiFcUpdRlVhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMuciINTH25x9BCc73E33EjCL9Lqov1TL7+QkgHe
T+J1hzwD8Mx2K+Lvvvu/aWkNrfMuNwyDUCiSI4D5QHa8T+F8fgN4OTpwYjirzel
5yoICMr9hVcbzDNv/oZKCxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viQKJ646s+
psiqXRytVvYInEhLvrJ0aV6zHfoigE/Bil6g/7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAXfeSMf77I+XTifigNna8x
t/ModjXr1fjF4pThEi5u6wsuRdfwjY2azEv3vevodTi4HoJReH6dFRa6y8c+UDgl
2iHiOKIpQqlbHEfQmHcD2fix+AaJKMnPgnku9qCFEMbgSRJpXz6fwnY1QuKE+i
R6ja0frUnt2jhiGG/F8RceXzohaaC/Cx7LUCUFwc0n7z32C9/Dtj7I1PMoacdzz
bjJzRKO/ZDv+UN/c9dwAk1l2AyPMwGbkuAy68EBstnIliW34aWm6IiHhxioVPKSp
VJfyiXP00EXqujthLAeChfjcns3I1YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEa
AYkCHwQYAQIAQCUCWrVJgwlbaAKCRC86dmkLVF4T+zD/9x/8APzgNJF3o3StrF
jvnV1ycyhWYGAeBjiu7wjsNWzMFov15tLjB7AqeVxZn+WKDD/mIOQ45OZvnYZuy
X7DR0JszAH9wrYTzXLVruAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbffy7B1HbvX/gYt
Rwe/uwdib10CagEzyX+2D3kT01HO5XThbXaNf8AN8ha91Jt2Q2UR2X5T6JcwttMz
FBvZn1LSmZyE0E0ehS2iUur4uWoPgpnuqVnbio1jbCvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmccliowYX9bT8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAULoqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXiKQ8DBWhBPVPruruLlaenTtZEosPc4I85yt5U9RcPTStcOr34sSw5yEaJagt6S
Gc5r9ysjkfH6+6rbilujxMgRROSqtqr+RyB+V9A5/OgtNzc8llK6u4UoOCde8jUUW
vqWKvJB/Kz3u4zaeNu2ZyyHa0qOuH+TETCw+jsy1lhEZqN5yQYGi4pVmDkY5vu
1XbJnbqPKpRxDgM9BecV9AMbPbgDq+5LnHJXg+G8YQ0g41R/hc1TEFdip5wM8AK
CwsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUoSARAar77kj7j2QR2SzEoS1FBvV7oS
mFeSNnz9xZssqrs6bTwSHM6YLDwc7Sdf2esDdyzONETWqrVCg+FxgL8hmo9hS4c
rR6tmrP0m0mpt+xLLskcaP7ogIXsyZnrEEsVw8PnfayoiPCdc3cMCR/1tnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLkvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4blD32WqvyxEs07vW6WBh7oqv3Zbm0yQrr8a6mDbpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSGm4U/G4JB+J1G1Ady+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmuOmhGyTss0G+3OcGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXikA01vE3/wgMqCxscbycbLjLb/bXiUFWo3rzoezeXjgi/DJx
jKBAyBTY05nMcTh109a0Fd9d0HbsOUDkIMnsgGBE766Piro6MHo0T0rX107Tp4pI
rwuSosc6XZCzdImj0Wc6axS/HeUKRxWdJXwno5awTwxkrJMXGfhCvSvbcC2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUBfygtH3eL5XZu28MiCpue8Y8GKJoBAUyvf
KeM1r08Jm3iRAC5a/DOAEQEAAYkEPgQYAQIAQCUCWrVlkjIbAgIpCRC86dmkLvf4
T8FdiAQZAQIABgvUCWrVlkjgAKCRDePL1hra+LjtHYD//MucxdFe6bx01dQR4tKhhQ
POLRqy6z1BY9ILCLowNdGZdqorogUiUymgn3VhEhVtxToohCn7q0uM01PnsRnOeS
EYjf8Xrb1clzkD6xULwmOclTb9bBxnBc/4PFvHAbZw3QzusaZniNgkuxt6BTflos
Of4in971kjmGK+T1zQ6mUMQug228NUQC+a84EPqYyAeY1sgvgB7hJbhYL0QAxhcW
6m20Rd8iEc6HyZJ3yCOCsKip/nRWAbf00vfHfRp0+m0ZwnJM8cPRFj0qqzFpKH9
HpDmTrC4wKP1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6vbMzKUMkUJK3
D3eHFAMkujmbfJmsMTJ0GPgn5SB1HyjCZnx6bh1b0yEUB9gKCMufaqXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOpH17T90m/PnUEEPwq8ksBhnMRgxa0RFidDP+n9fgtv
HLmrOqXzBCVXh0mdWYlrWvmzQFWzG7Aoe55fkf8nAEpsalrCdtANUBHRXA00QxG
AHModJQqVBSmqMvuAdjukDWPfu5y0My5ddu+hiUzQyLjL5Hhd5LOUDDewlZgIw1j
xrEAuzDKetnem8GkHxDgg8koev5frmShJu7vSjpkCng3EIJSggMOPFjJuLwtz
vjHeDnbJy6uNL65ckJy6whGjeADS2WA1D6Tfekkc21ssIxk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8Me1n9qr5KcKQlmfdfBNEyyceBhyv10MDyHOKC+7PofMtkGbq
13QieRHv5GJ8LB3fc1qHv8pwTT03Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279ioVTrwpECse0XkiRykToTjwOb73CGkBZzpjyqux/rmCV/fp4ALdSW8zbz
FJVORaihovWwzjpFQKhwcU91ABxi2UvVm14v0Afei7oiJPSU1zM4fEny4oiIBX1R
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMi+mWra23EwjChaxpvjjcuh
5illC5Zq781aCYRygYQw+hu5nfkoh1R+Z50Ubxjd/aqUfnGIAx7kPMD3Lof4K1dD
Q8ppQriUvxVo+4nPv6rpTy/PyqCLWDjkguHpsEfsmkwajrAz0QNSAU5CJ0G2Zu4
yxyvYlumHCE17nbFrm0vIia75Sa8KnywTdsyzsu3Xcofc3g+g1xWtpjJqy2bYX1lqz
9uDOWtArWHOis6bq819RE6xr1RBVXS6uqgQIZFBGyq66b0d1q4D2JdsUvgEMahbc

```
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZYlNr3lydh+dFHIEkH53HzQe6l88HEic
+0jVnLkCDQRa55wJARAAYlya2Lx6gyoWoJN1a6740q3o8e9d4KggQofGMTCflmeq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNjyjmNT1gAgrhPNB4HtBXUm2ws57WK
DNmade914L7FWTPAWBG2Wn448OEHTqsClICXXWy9IIcglAEYIq0Yq5mAdTEgRJS
Z8t4GpwtdL9gNQyFXaWqmDmkAsCygQMvhAlmu9xOIZQG5CxSnZFk7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUidgfPCSvOUW1JojsdCQA
L+RZJcoXq7lfOFj/eNjeOSstCTDPfTCL+kThE6E5neDtbQHBYkEX1BRiTedsV4+M
ucgiTrdQFWKF89G72xdv8ut9AYYQ2bEYU+JAYhUH8rYYui2dHKJ1gjNvJscuUWb
+QEeqJIRleJRhr0+/CHgMs4fZAkWF1VFhKBkcKmEjLn1f7EJJUUW84ZhKXjO/AUPX
1CHsNjz1RceuJCJYoxlcwsq6jTE50GiNzcIxTn9xUc0UMKfeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm91hkGwqTZg0lRx5eqPxF+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6SYKqroYwBGGz6MJs5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9z1W8qsA
EQEEAAyKEPgQYAQgACQUCWuecCQIBAgIpCRC86dmkLVR4T8FdIAQZAQgABgUCWuec
COAKCRBQ3szEcQ5hr+ykD/4tOLRHFXuKUCxgGaUbUcvtsFrwBKnalcYjqapMs8u
6Sk0wfGR132G/GhOrp0Ts/M0kbObq6VLTh8N5Yc/53ME18zQFw9Y5AmRow4PZXER
ujss5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEfHYT8UTxwnGmXOvNkMM7md
xD3CV5q6VAt8WKBo/220II3fcQlc9r/oWx4kXXkb0v9hoGwKbDJ1tzqTPrp/xFt
yohqnvImpnlz+Q9zXmbrWYL9/g8VCmW>NN2gju2G3Lu/T1FUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CML1S57KSgCo5HUhQWeSNHI+fpe5oX6FALPT9JLDce80Zz1i
cZZ0MELP37mOOQun0AlmHm/hvzf0f311PtzbzqWaE51tJvgUR/nZFo6Ta305Ezs
3V1EJNQ1Ijf/6DH87SxvAoRIARCuZd0qxBcDKoavpFzUtbJd241RA3WJpkEiMqKv
RDVZkE4b6TW61f0o+LaVfK6E8oLpixegS4fiqC16mFrOdyRk+RJJfIUyz0WTDVm
g0U1C01ezokMSqkJ7724pyjr2xf/r9/sC6aOJwb/1kgZkJfC6NqL7TlxVA31dUga
LEOvEJTT4gl+tYtfscdvAlCtqL0jduskuo+RXcB1tmXhA+tShW0pbS2Rtx/ixua
KohVD/0R4QxiSwQmICNm9mw9ydi1lyjYXX5a9x4wMJracNY/LBybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6M1I2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBXuAX7+HgPPSFtrHQONCALxxz1bNpS+zxt9r0MiLgcLyspWxSdmoYGZ6nQP
R05Nm/ZVS+u2imPCRzNUZEMa+d1E6khx0rS0dPiuj407NtPeYDKkoQtNagspsDvh
ck7CSqAiKMq06UBTxqlTSRkm62eOtc3p30eHu5GRZF1uzTET0ZxYkaPgdrQknx
ozjP5mC7x+451cCfmct94TFNL5HwEUJVpmOgmzILC18yoDTWz1oo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmN3bCLuc/jq7ysGq69xiKmTlUeXFm+aojcRO5i
zyShIRJZOGZfuzDYFDbMV9ama/YQGygLw//zP5ju5SW26dNx1f3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRGLU/LpNSefnvDFTtEIRcpOHc
bhayG0bk51Bd4mioOXNsKy4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj20+x+j
qx2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==
=Gi5D
-----END PGP PUBLIC KEY BLOCK-----
```

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

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

- Import the Amazon ECS PGP public key with the following command.

```
gpg --import <public_key_filename>
```

- Download the Amazon ECS CLI signatures. The signatures are ASCII detached PGP signatures stored in files with the extension .asc. The signatures file has the same name as its corresponding executable, with .asc appended.
 - For macOS systems:

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

- For Linux systems:

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

- For Windows systems:

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

5. Verify the signature.

- For macOS and Linux systems:

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

- For Windows systems:

```
PS C:\> gpg --verify ecs-cli.asc 'C:\Program Files\Amazon\ECSCLI\ecs-cli.exe'
```

Expected output:

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

Important

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

Step 3: Apply Execute Permissions to the Binary

Apply execute permissions to the binary.

- For macOS and Linux systems:

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

- For Windows systems:

Edit the environment variables and add C:\Program Files\Amazon\ECSCLI to the PATH variable field, separated from existing entries by using a semicolon. For example:

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

Restart PowerShell (or the command prompt) so the changes go into effect.

Note

Once the PATH variable is set, the Amazon ECS CLI can be used from either Windows PowerShell or the command prompt.

Step 4: Complete the Installation

Verify that the CLI is working properly.

```
ecs-cli --version
```

Proceed to [Configuring the Amazon ECS CLI \(p. 60\)](#).

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 \(p. 49\)](#).

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 \(p. 62\)](#). 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 \(p. 55\)](#).
- Install and configure the AWS CLI. For more information, see [AWS Command Line Interface](#).

Step 1: Create the Task Execution IAM Role

The Amazon ECS container agent makes calls to AWS APIs on your behalf, so it requires an IAM policy and role for the service to know that the agent belongs to you. This IAM role is referred to as a task execution IAM role. If you already have a task execution role created to use, you can skip this step. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

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 \(p. 60\)](#).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration, which defines the AWS region to use, resource creation prefixes, and the cluster name to use with the Amazon ECS CLI:

```
ecs-cli configure --cluster tutorial --default-launch-type FARGATE --config-name tutorial --region us-west-2
```

2. Create a CLI profile using your access key and secret key:

```
ecs-cli configure profile --access-key AWS_ACCESS_KEY_ID --secret-key AWS_SECRET_ACCESS_KEY --profile-name tutorial-profile
```

Step 3: Create a Cluster and Configure the Security Group

To create an ECS cluster and security group

1. Create an Amazon ECS cluster with the `ecs-cli up` command. Because you specified Fargate as your default launch type in the cluster configuration, this command creates an empty cluster and a VPC configured with two public subnets.

```
ecs-cli up --cluster-config tutorial --ecs-profile tutorial-profile
```

This command may take a few minutes to complete as your resources are created. The output of this command contains the VPC and subnet IDs that are created. Take note of these IDs as they are used later.

2. Using the AWS CLI, retrieve the default security group ID for the VPC. Use the VPC ID from the previous output:

```
aws ec2 describe-security-groups --filters Name=vpc-id,Values=VPC_ID --region us-west-2
```

The output of this command contains your security group ID, which is used in the next step.

3. Using AWS CLI, add a security group rule to allow inbound access on port 80:

```
aws ec2 authorize-security-group-ingress --group-id security_group_id --protocol tcp --port 80 --cidr 0.0.0.0/0 --region us-west-2
```

Step 4: Create a Compose File

For this step, create a simple Docker compose file that creates a simple PHP web application. At this time, the Amazon ECS CLI supports [Docker compose file syntax](#) versions 1, 2, and 3. This tutorial uses Docker compose v3.

Here is the compose file, which you can name `docker-compose.yml`. The web container exposes port 80 for inbound traffic to the web server. It also configures container logs to go to the CloudWatch log group created earlier. This is the recommended best practice for Fargate tasks.

```
version: '3'
services:
  web:
    image: amazon/amazon-ecs-sample
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
```

```
awslogs-region: us-west-2
awslogs-stream-prefix: web
```

Note

If your account already contains a CloudWatch Logs log group named `tutorial` in the `us-west-2` Region, choose a unique name so the ECS CLI creates a new log group for this tutorial.

In addition to the Docker compose information, there are some parameters specific to Amazon ECS that you must specify for the service. Using the VPC, subnet, and security group IDs from the previous step, create a file named `ecs-params.yml` with the following content:

```
version: 1
task_definition:
  task_execution_role: ecsTaskExecutionRole
  ecs_network_mode: awsvpc
  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	Health	State	Ports
TaskDefinition			
tutorial/0c2862e6e39e4eff92ca3e4f843c5b9a/web	UNKNOWN	RUNNING	34.222.202.55:80->80/tcp
tutorial:1	UNKNOWN		

In the above example, you can see the web container from your compose file, and also the IP address and port of the web server. If you point your web browser at that address, you should see the PHP web application. Also in the output is the task-id value for the container. Copy the task ID as you use it in the next step.

Step 7: View the Container Logs

View the logs for the task:

```
ecs-cli logs --task-id 0c2862e6e39e4eff92ca3e4f843c5b9a --follow --cluster-config tutorial  
--ecs-profile tutorial-profile
```

Note

The --follow option tells the Amazon ECS CLI to continuously poll for logs.

Step 8: Scale the Tasks on the Cluster

You can scale up your task count to increase the number of instances of your application with `ecs-cli compose service scale`. In this example, the running count of the application is increased to two.

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

Now you should see two more containers in your cluster:

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

Output:

Name	Health	State	Ports
tutorial/0c2862e6e39e4eff92ca3e4f843c5b9a/web	UNKNOWN	RUNNING	34.222.202.55:80->80/tcp
tutorial/d9fbcbc931d2e47ae928fcf433041648f/web	UNKNOWN	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.

Simple PHP App

Congratulations

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

The container is running PHP version 5.3.10-1ubuntu3.15.

Step 10: Clean Up

When you are done with this tutorial, you should clean up your resources so they do not incur any more charges. First, delete the service so that it stops the existing containers and does not try to run any more tasks.

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

Now, take down your cluster, which cleans up the resources that you created earlier with ecs-cli up.

```
ecs-cli down --force --cluster-config tutorial --ecs-profile tutorial-profile
```

Tutorial: Creating a Cluster with an EC2 Task Using the Amazon ECS CLI

This tutorial shows you how to set up a cluster and deploy a task using the EC2 launch type.

Prerequisites

Complete the following prerequisites:

- Complete the steps in [Setting up with Amazon ECS \(p. 7\)](#) and verify that your AWS user has either the permissions specified in the [AdministratorAccess](#) or the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.
- Install the Amazon ECS CLI. For more information, see [Installing the Amazon ECS CLI \(p. 55\)](#).
- Install and configure the AWS CLI. For more information, see [AWS Command Line Interface](#).

Step 1: Configure the Amazon ECS CLI

Before you can start this tutorial, you must install and configure the Amazon ECS CLI. For more information, see [Installing the Amazon ECS CLI \(p. 55\)](#).

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 \(p. 60\)](#).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration:

```
ecs-cli configure --cluster ec2-tutorial --default-launch-type EC2 --config-name ec2-tutorial --region us-west-2
```

2. Create a profile using your access key and secret key:

```
ecs-cli configure profile --access-key AWS_ACCESS_KEY_ID --secret-key AWS_SECRET_ACCESS_KEY --profile-name ec2-tutorial-profile
```

Step 2: Create Your Cluster

The first action you should take is to create a cluster of Amazon ECS container instances that you can launch your containers on with the `ecs-cli up` command. There are many options that you can choose to configure your cluster with this command, but most of them are optional. In this example, you create a simple cluster of two `t2.medium` container instances that use the `id_rsa` key pair for SSH access (substitute your own key pair here).

By default, the security group created for your container instances opens port 80 for inbound traffic. You can use the `--port` option to specify a different port to open, or if you have more complicated security group requirements, you can specify an existing security group to use with the `--security-group` option.

```
ecs-cli up --keypair id_rsa --capability-iam --size 2 --instance-type t2.medium --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

This command may take a few minutes to complete as your resources are created. Now that you have a cluster, you can create a Docker compose file and deploy it.

Step 3: Create a Compose File

For this step, create a simple Docker compose file that creates a simple PHP web application. At this time, the Amazon ECS CLI supports [Docker compose file syntax](#) versions 1, 2, and 3. This tutorial uses Docker Compose version 3.

Here is the compose file, which you can call `docker-compose.yml`. The `web` container exposes port 80 to the container instance for inbound traffic to the web server. A logging configuration for the containers is also defined.

```
version: '3'
services:
  web:
    image: amazon/amazon-ecs-sample
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: ec2-tutorial
        awslogs-region: us-west-2
        awslogs-stream-prefix: web
```

When using Docker Compose version 3 format, the CPU and memory specifications must be specified separately. Create a file named `ecs-params.yml` with the following content:

```
version: 1
task_definition:
  services:
    web:
      cpu_shares: 100
      mem_limit: 524288000
```

Step 4: Deploy the Compose File to a Cluster

After you create the compose file, you can deploy it to your cluster with the `ecs-cli compose up` command. By default, the command looks for a compose file called `docker-compose.yml` and an optional ECS parameters file called `ecs-params.yml` in the current directory, but you can specify

a different file with the `--file` option. By default, the resources created by this command have the current directory in the title, but you can override that with the `--project-name project_name` option. The `--create-log-groups` option creates the CloudWatch log groups for the container logs.

```
ecs-cli compose up --create-log-groups --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Step 5: View the Running Containers on a Cluster

After you deploy the compose file, you can view the containers that are running on your cluster with the `ecs-cli ps` command.

```
ecs-cli ps --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Output:

Name	State	Ports
TaskDefinition Health ec2-tutorial/53c943778bf048ce954a6cb96425adeb/web	RUNNING	54.201.208.32:80->80/tcp
ecscompose:1 UNKNOWN		

In the above example, you can see the web container from your compose file, and also the IP address and port of the web server. If you point a web browser to that address, you should see the PHP web application.

Step 6: Scale the Tasks on a Cluster

You can scale your task count up so you could have more instances of your application with the `ecs-cli compose scale` command. In this example, you can increase the count of your application to two.

```
ecs-cli compose scale 2 --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Now you should see two more containers in your cluster:

```
ecs-cli ps --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Output:

Name	State	Ports
TaskDefinition Health ec2-tutorial/53c943778bf048ce954a6cb96425adeb/web	RUNNING	54.201.208.32:80->80/tcp
ecscompose:1 UNKNOWN		
ec2-tutorial/9451480d53534a129fe6794941ad63dc/web	RUNNING	52.43.118.109:80->80/tcp
ecscompose:1 UNKNOWN		

Step 7: Create an ECS Service from a Compose File

Now that you know that your containers work properly, you can make sure that they are replaced if they fail or stop. You can do this by creating a service from your compose file with the `ecs-cli compose service up` command. This command creates a task definition from the latest compose file (if it does not already exist) and creates an ECS service with it, with a desired count of 1.

Before starting your service, stop the containers from your compose file with the `ecs-cli compose down` command so that you have an empty cluster to work with.

```
ecs-cli compose down --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Now you can create your service.

```
ecs-cli compose service up --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Step 8: View your Web Application

Enter the IP address for the task in your web browser and you should see a webpage that displays the **Simple PHP App** web application.

Simple PHP App

Congratulations

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

The container is running PHP version 5.3.10-1ubuntu3.15.

Step 9: Clean Up

When you are done with this tutorial, you should clean up your resources so they do not incur any more charges. First, delete the service so that it stops the existing containers and does not try to run any more tasks.

```
ecs-cli compose service rm --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Now, take down your cluster, which cleans up the resources that you created earlier with **ecs-cli up**.

```
ecs-cli down --force --cluster-config ec2-tutorial --ecs-profile ec2-tutorial-profile
```

Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI

This tutorial shows a simple walkthrough of creating an Amazon ECS service that is configured to use service discovery. Many of the service discovery configuration values can be specified with either the ECS parameters file or flags. When flags are used, they take precedence over the ECS parameters file if both are present. When using the Amazon ECS CLI, the compose project name is used as the name for your ECS service.

Prerequisites

It is expected that you have completed the following prerequisites before continuing on:

- Set up an AWS account.
- Install the Amazon ECS CLI. For more information, see [Installing the Amazon ECS CLI \(p. 55\)](#).
- Create a VPC. For more information, see [the section called “Create a virtual private cloud” \(p. 11\)](#).

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 \(p. 55\)](#).

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 \(p. 60\)](#).

To create an Amazon ECS CLI configuration

1. Create a cluster configuration:

```
ecs-cli configure --cluster ec2-tutorial --region us-east-1 --default-launch-type EC2  
--config-name ec2-tutorial
```

2. Create a profile using your access key and secret key:

```
ecs-cli configure profile --access-key AWS_ACCESS_KEY_ID --secret-  
key AWS_SECRET_ACCESS_KEY --profile-name ec2-tutorial
```

Note

If this is the first time that you are configuring the Amazon ECS CLI, these configurations are marked as default. If this is not your first time configuring the Amazon ECS CLI, see the [Amazon ECS Command Line Reference](#) in the *Amazon Elastic Container Service Developer Guide* to set this as the default configuration and profile.

Create an Amazon ECS Service Configured to Use Service Discovery

Use the following steps to create an Amazon ECS service that is configured to use service discovery with the Amazon ECS CLI.

To create an Amazon ECS service configured to use service discovery

1. Create an Amazon ECS service named `backend` and create a private DNS namespace named `tutorial` within a VPC. In this example, the task is using the `awsvpc` network mode, so the `container_name` and `container_port` values are not required.

```
ecs-cli compose --project-name backend service up --private-dns-namespace tutorial --  
vpc vpc-04deee8176dce7d7d --enable-service-discovery
```

Output:

```
INFO[0001] Using ECS task definition TaskDefinition="backend:1"  
INFO[0002] Waiting for the private DNS namespace to be created...  
INFO[0002] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS  
WARN[0033] Defaulting DNS Type to A because network mode was awsvpc  
INFO[0033] Waiting for the Service Discovery Service to be created...  
INFO[0034] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS  
INFO[0065] Created an ECS service service=backend  
taskDefinition="backend:1"  
INFO[0066] Updated ECS service successfully desiredCount=1  
serviceName=backend  
INFO[0081] (service backend) has started 1 tasks: (task 824b5a76-8f9c-4beb-  
a64b-6904e320630e). timestamp="2018-09-12 00:00:26 +0000 UTC"
```

```
INFO[0157] Service status           desiredCount=1 runningCount=1
  serviceName=backend
INFO[0157] ECS Service has reached a stable state   desiredCount=1 runningCount=1
  serviceName=backend
```

2. Create another service named `frontend` in the same private DNS namespace. Because the namespace already exists, the Amazon ECS CLI uses it instead of creating a new one.

```
ecs-cli compose --project-name frontend service up --private-dns-namespace tutorial --
vpc vpc-04deee8176dce7d7d --enable-service-discovery
```

Output:

```
INFO[0001] Using ECS task definition           TaskDefinition="frontend:1"
INFO[0002] Using existing namespace ns-kvhnzhb5vxplfmls
WARN[0033] Defaulting DNS Type to A because network mode was awsvpc
INFO[0033] Waiting for the Service Discovery Service to be created...
INFO[0034] Cloudformation stack status          stackStatus=CREATE_IN_PROGRESS
INFO[0065] Created an ECS service             service=frontend
  taskDefinition="frontend:1"
INFO[0066] Updated ECS service successfully    desiredCount=1
  serviceName=frontend
INFO[0081] (service frontend) has started 1 tasks: (task 824b5a76-8f9c-4beb-
a64b-6904e320630e). timestamp="2018-09-12 00:00:26 +0000 UTC"
INFO[0157] Service status                     desiredCount=1 runningCount=1
  serviceName=frontend
INFO[0157] ECS Service has reached a stable state   desiredCount=1 runningCount=1
  serviceName=frontend
```

3. Verify that the two services are able to discover each other within the VPC using DNS. The DNS hostname uses the following format: `<service_discovery_service_name>. <service_discovery_namespace>`. For this example, the `frontend` service can be discovered at `frontend.tutorial` and the `backend` service can be discovered at `backend.tutorial`. Because these are private DNS namespaces, these DNS names only resolve when within the specified VPC.
4. To update the service discovery settings, update the settings for the `frontend` service. The values that can be updated are the DNS TTL and the value for the health check custom config failure threshold.

```
ecs-cli compose --project-name frontend service up --update-service-discovery --dns-
type SRV --dns-ttl 120 --healthcheck-custom-config-failure-threshold 2
```

Output:

```
INFO[0001] Using ECS task definition           TaskDefinition="frontend:1"
INFO[0001] Updated ECS service successfully    desiredCount=1
  serviceName=frontend
INFO[0001] Service status                     desiredCount=1 runningCount=1
  serviceName=frontend
INFO[0001] ECS Service has reached a stable state   desiredCount=1 runningCount=1
  serviceName=frontend
INFO[0002] Waiting for your Service Discovery resources to be updated...
INFO[0002] Cloudformation stack status          stackStatus=UPDATE_IN_PROGRESS
```

5. To clean up, delete the Amazon ECS service and the service discovery resources. When the `frontend` service is deleted, the Amazon ECS CLI automatically removes the associated service discovery service.

```
ecs-cli compose --project-name frontend service rm
```

```
INFO[0000] Updated ECS service successfully          desiredCount=0
serviceName=frontend
INFO[0001] Service status                         desiredCount=0 runningCount=1
serviceName=frontend
INFO[0016] Service status                         desiredCount=0 runningCount=0
serviceName=frontend
INFO[0016] (service frontend) has stopped 1 running tasks: (task 824b5a76-8f9c-4beb-
a64b-6904e320630e). timestamp="2018-09-12 00:37:25 +0000 UTC"
INFO[0016] ECS Service has reached a stable state   desiredCount=0 runningCount=0
serviceName=frontend
INFO[0016] Deleted ECS service                     service=frontend
INFO[0016] ECS Service has reached a stable state   desiredCount=0 runningCount=0
serviceName=frontend
INFO[0027] Waiting for your Service Discovery Service resource to be deleted...
INFO[0027] Cloudformation stack status             stackStatus=DELETE_IN_PROGRESS
```

6. To complete the cleanup, delete the backend service along with the private DNS namespace that was created with it. The Amazon ECS CLI associates the AWS CloudFormation stack for the private DNS namespace with the Amazon ECS service for which it was created. When the service is deleted, the namespace is also deleted.

```
ecs-cli compose --project-name backend service rm --delete-namespace
```

Amazon ECS command line reference

The following commands are available in the Amazon ECS CLI. Help text for each command is available by appending the `--help` option to the final command argument. List the help text for the Amazon ECS CLI by using the following command:

```
ecs-cli --help
```

Note

Ensure that you are using the latest version of the Amazon ECS CLI. The latest version is 1.17.0. For release notes, see [Changelog](#).

Available commands

- [ecs-cli \(p. 74\)](#)
- [ecs-cli configure \(p. 75\)](#)
- [ecs-cli up \(p. 81\)](#)
- [ecs-cli down \(p. 89\)](#)
- [ecs-cli scale \(p. 91\)](#)
- [ecs-cli ps \(p. 93\)](#)
- [ecs-cli push \(p. 94\)](#)
- [ecs-cli pull \(p. 96\)](#)
- [ecs-cli images \(p. 98\)](#)
- [ecs-cli license \(p. 101\)](#)
- [ecs-cli compose \(p. 102\)](#)
- [ecs-cli compose service \(p. 114\)](#)
- [ecs-cli logs \(p. 137\)](#)

- [ecs-cli check-attributes \(p. 139\)](#)
- [ecs-cli registry-creds \(p. 140\)](#)
- [ecs-cli local \(p. 146\)](#)
- [Using Docker Compose File Syntax \(p. 153\)](#)
- [Using Amazon ECS Parameters \(p. 155\)](#)

ecs-cli

Description

The Amazon ECS command line interface (CLI) provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment. The Amazon ECS CLI supports [Docker Compose](#), a popular open-source tool for defining and running multi-container applications.

For a quick walkthrough of the Amazon ECS CLI, see the [Tutorial: Creating a cluster with a Fargate task using the Amazon ECS CLI \(p. 62\)](#).

Help text is available for each individual subcommand with **ecs-cli *subcommand* --help**.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli [--version] [subcommand] [--help]
```

Options

Name	Description
--version, -v	Prints the version information for the Amazon ECS CLI. Required: No
--help, -h	Show the help text for the specified command. Required: No

Available Subcommands

The **ecs-cli** command supports the following subcommands:

configure

Configures your AWS credentials, the Region to use, and the ECS cluster name to use with the Amazon ECS CLI. For more information, see [ecs-cli configure \(p. 75\)](#).

migrate

Migrates a legacy configuration file (ECS CLI v0.6.6 and older) to the new configuration file format (ECS CLI v1.0.0 and later). The command prints a summary of the changes to be made and then asks for confirmation to proceed. For more information, see [ecs-cli configure migrate \(p. 81\)](#).

up

Creates the Amazon ECS cluster (if it does not already exist) and the AWS resources required to set up the cluster. For more information, see [ecs-cli up \(p. 81\)](#).

down

Deletes the AWS CloudFormation stack that was created by **ecs-cli up** and the associated resources. For more information, see [ecs-cli down \(p. 89\)](#).

scale

Modifies the number of container instances in an Amazon ECS cluster. For more information, see [ecs-cli scale \(p. 91\)](#).

logs

Retrieves container logs from CloudWatch Logs. Only valid for tasks that use the `awslogs` driver and has a log stream prefix specified. For more information, see [ecs-cli logs \(p. 137\)](#).

ps

Lists all of the running containers in an Amazon ECS cluster. For more information, see [ecs-cli ps \(p. 93\)](#).

push

Pushes an image to an Amazon ECR repository. For more information, see [ecs-cli push \(p. 94\)](#).

pull

Pulls an image from an ECR repository. For more information, see [ecs-cli pull \(p. 96\)](#).

images

Lists all of the running containers in an ECS cluster. For more information, see [ecs-cli images \(p. 98\)](#).

license

Prints the `LICENSE` files for the Amazon ECS CLI and its dependencies. For more information, see [ecs-cli license \(p. 101\)](#).

compose

Executes `docker-compose`-style commands on an ECS cluster. For more information, see [ecs-cli compose \(p. 102\)](#).

help

Shows the help text for the specified command.

ecs-cli configure

Configures the AWS Region to use, resource creation prefixes, and the Amazon ECS cluster name to use with the Amazon ECS CLI. Stores a single named cluster configuration in the `~/.ecs/config` file. The first cluster configuration that is created is set as the default.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Working with Multiple Cluster Configurations

The following should be noted when using multiple cluster configurations:

- Multiple cluster configurations may be stored, but one is always the default.

- The first cluster configuration that is stored is set as the default.
- Use the `ecs-cli configure default` command to change which cluster configuration is set as the default. For more information, see [ecs-cli configure default \(p. 77\)](#).
- A non-default cluster configuration can be referenced in a command by using the `--cluster-config` flag.

For more information, see [ecs-cli configure default \(p. 77\)](#).

Note

Ensure that you are using the latest version of the Amazon ECS CLI to use all configuration options.

Syntax

```
ecs-cli configure --cluster cluster_name --region region [--config-name config_name] [--cfn-stack-name stack_name] [--default-launch-type launch_type] [--help]
```

Options

Name	Description
<code>--cluster, -c <i>cluster_name</i></code>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</p> <p>Type: String</p> <p>Required: Yes</p>
<code>--region, -r <i>region</i></code>	<p>Specifies the AWS Region to use. Defaults to the Region configured using either the <code>ecs-cli configure</code> or <code>aws configure</code> commands.</p> <p>Type: String</p> <p>Required: Yes</p>
<code>--config-name <i>config_name</i></code>	<p>Specifies the name of this cluster configuration. This is the name that can be referenced in commands using the <code>--cluster-config</code> flag. If this option is omitted, then the name is set to <code>default</code>.</p> <p>Type: String</p> <p>Required: No</p>
<code>--cfn-stack-name <i>stack_name</i></code>	<p>Specifies the stack name to add to the AWS CloudFormation stack that is created on <code>ecs-cli up</code>.</p> <p>Important It is not recommended to use this parameter. It is included to ensure backwards compatibility with previous versions of the ECS CLI.</p> <p>Type: String</p> <p>Default: <code>amazon-ecs-cli-setup-<cluster_name></code></p> <p>Required: No</p>

Name	Description
--default-launch-type <i>launch_type</i>	<p>Specifies the default launch type to use. Valid values are FARGATE or EC2. If not specified, no default launch type is used. For more information about launch types, see Amazon ECS launch types (p. 247).</p> <p>Type: String</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Examples

Example

This example configures the Amazon ECS CLI to create a cluster configuration named `ecs-cli-demo`, which uses FARGATE as the default launch type for cluster `ecs-cli-demo` in the `us-east-1` region.

```
ecs-cli configure --region us-east-1 --cluster ecs-cli-demo --default-launch-type FARGATE
--config-name ecs-cli-demo
```

Output:

```
INFO[0000] Saved ECS CLI cluster configuration ecs-cli-demo.
```

Contents of the `~/.ecs/config` file after running the command:

```
version: v1
default: ecs-cli-demo
clusters:
  ecs-cli-demo:
    cluster: ecs-cli-demo
    region: us-east-1
    default_launch_type: FARGATE
```

ecs-cli configure default

Sets the cluster configuration to be read from by default.

Note

Unlike the AWS CLI, the Amazon ECS CLI does not expect or require that the default configuration be named `default`. The name of a configuration does not determine whether it is default.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli configure default --config-name config_name [--help]
```

Options

Name	Description
--config-name <i>config_name</i>	Specifies the name of the cluster configuration to use by default in subsequent commands. Type: String Required: Yes
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example

This example configures the Amazon ECS CLI to set the `ecs-cli-demo` cluster configuration as the default.

```
ecs-cli configure default --config-name ecs-cli-demo
```

There is no output if the command is successful.

ecs-cli configure profile

Configures your AWS credentials in a named Amazon ECS profile, which is stored in the `~/.ecs/credentials` file. If multiple profiles are created, you can change the profile used by default with the **ecs-cli configure profile default** command. For more information, see [ecs-cli configure profile default \(p. 80\)](#).

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

You can configure your AWS credentials in several ways:

- You can set the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables. When you run **ecs-cli configure profile**, the values of those variables are stored in the Amazon ECS CLI configuration file.
- You can pass credentials directly on the command line with the `--access-key`, `--secret-key`, and `--session-token` options.
- You can provide the name of a new profile with the `--profile-name` flag. If a profile name is not provided, then the profile is named `default`.
- The first profile configured is set as the `default` profile. The Amazon ECS CLI uses credentials specified in this profile unless the `--ecs-profile` flag is used.

Working with Multiple Profiles

The following should be noted when using multiple profiles:

- Multiple profiles may be configured, but one is always the `default`. This profile is used when an Amazon ECS CLI command is run that requires credentials.

- The first profile that is created is set as the default profile.
- To change the default profile, use the `ecs-cli configure profile default` command. For more information, see [ecs-cli configure profile default \(p. 80\)](#).
- A non-default profile can be referenced in a command using the `--ecs-profile` flag.

Syntax

```
ecs-cli configure profile --profile-name profile_name --access-key aws_access_key_id --secret-key aws_secret_access_key [--session-token token] [--help]
```

Options

Name	Description
<code>--profile-name <i>profile_name</i></code>	<p>Specifies the name of this ECS profile. This is the name that can be referenced in commands using the <code>--ecs-profile</code> flag. If this option is omitted, then the name is set to <code>default</code>.</p> <p>Type: String</p> <p>Required: Yes</p>
<code>--access-key <i>aws_access_key_id</i></code>	<p>Specifies the AWS access key to use. If the <code>AWS_ACCESS_KEY_ID</code> environment variable is set when <code>ecs-cli configure profile</code> is run, then the AWS access key ID is set to the value of that environment variable.</p> <p>Type: String</p> <p>Required: Yes</p>
<code>--secret-key <i>aws_secret_access_key</i></code>	<p>Specifies the AWS secret key to use. If the <code>AWS_SECRET_ACCESS_KEY</code> environment variable is set when <code>ecs-cli configure profile</code> is run, then the AWS secret access key is set to the value of that environment variable.</p> <p>Type: String</p> <p>Required: Yes</p>
<code>--session-token <i>token</i></code>	<p>Specifies the AWS session token to use. If the <code>AWS_SESSION_TOKEN</code> environment variable is set when <code>ecs-cli configure profile</code> is run, then the AWS session token is set to the value of that environment variable. For more information about using a session token for temporary access, see Requesting Temporary Security Credentials.</p> <p>Type: String</p> <p>Required: No</p>
<code>--help, -h</code>	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Examples

Example 1

This example configures the Amazon ECS CLI to create and use a profile named default with a set of access keys.

```
ecs-cli configure profile --profile-name default --access-key $AWS_ACCESS_KEY_ID --secret-key $AWS_SECRET_ACCESS_KEY
```

Output:

```
INFO[0000] Saved ECS CLI profile configuration default.
```

Example 2

This example configures the Amazon ECS CLI to create and use a profile named default with a set of access keys and an AWS session token.

```
ecs-cli configure profile --profile-name default --access-key $AWS_ACCESS_KEY_ID --secret-key $AWS_SECRET_ACCESS_KEY --session-token $AWS_SESSION_TOKEN
```

Output:

```
INFO[0000] Saved ECS CLI profile configuration default.
```

ecs-cli configure profile default

Sets the Amazon ECS profile to be read from by default.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli configure profile default --profile-name profile_name [--help]
```

Options

Name	Description
--profile-name <i>profile_name</i>	Specifies the name of the ECS profile to be marked as default. Type: String Required: Yes
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example

This example configures the Amazon ECS CLI to set the `default` profile as the default profile to be used.

```
ecs-cli configure profile default --profile-name default
```

There is no output if the command is successful.

ecs-cli configure migrate

Migrates a legacy configuration file (ECS CLI v0.6.6 and older) to the new configuration file format (ECS CLI v1.0.0 and later). The command prints a summary of the changes to be made and then asks for confirmation to proceed.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli configure migrate [--force] [--help]
```

Options

Name	Description
<code>--force</code>	Omits the interactive description and confirmation step that normally occurs during the configuration file migration. Required: No
<code>--help</code> , <code>-h</code>	Shows the help text for the specified command. Required: No

Examples

Example

This example migrates the legacy Amazon ECS CLI configuration file to the new YAML format.

```
ecs-cli configure migrate
```

ecs-cli up

Creates the Amazon ECS cluster (if it does not already exist) and the AWS resources required to set up the cluster.

This command creates a new AWS CloudFormation stack called `amazon-ecs-cli-setup-cluster_name`. You can view the progress of the stack creation in the AWS Management Console.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli up [--capability-iam | --instance-role instance-profile-name] [--keypair keypair_name] [--size n] [--azs availability_zone_1,availability_zone_2] [--security-group security_group_id[,security_group_id,...]] [--cidr ip_range] [--port port_number] [--subnets subnet_1,subnet_2] [--vpc vpc_id] [--extra-user-data string] [--instance-type instance_type] [--image-id ami_id] [--launch-type launch_type] [--no-associate-public-ip-address] [--force] [--tags key1=value1,key2=value2] [--cluster cluster_name] [--region region] [--empty] [--verbose] [--help]
```

Options

Name	Description
--capability-iam	<p>Acknowledges that this command may create IAM resources.</p> <p>Note This parameter is only supported with tasks that use the EC2 launch type.</p> <p>This parameter is required if you do not specify an instance profile name with --instance-role. You cannot specify both options.</p> <p>Required: No</p>
--keypair <i>keypair_name</i>	<p>Specifies the name of an existing Amazon EC2 key pair to enable SSH access to the EC2 instances in your cluster.</p> <p>Note This parameter is only supported with tasks that use the EC2 launch type.</p> <p>For more information about creating a key pair, see Setting Up with Amazon EC2 in the <i>Amazon EC2 User Guide for Linux Instances</i>.</p> <p>Type: String</p> <p>Required: No</p>
--size <i>n</i>	<p>Specifies the number of instances to launch and register to the cluster.</p> <p>Note This parameter is only supported with tasks that use the EC2 launch type.</p> <p>Type: Integer</p> <p>Default: 1</p> <p>Required: No</p>
--azs <i>availability_zone_1,availability_zone_2</i>	<p>Specifies a comma-separated list of two VPC Availability Zones in which to create subnets (these zones must have the available status). We recommend this option if you do not specify a VPC ID with the --vpc option.</p>

Name	Description
	<p>Warning Leaving this option blank can result in a failure to launch container instances when the randomly chosen zone is unavailable.</p> <p>Type: String</p> <p>Required: No</p>
--security-group <i>security_group_id</i> [, <i>security_group_id</i>]	<p>Specifies a comma-separated list of existing security groups to associate with your container instances. If you do not specify a security group here, then a new one is created.</p> <p>For more information, see Security Groups in the <i>Amazon EC2 User Guide for Linux Instances</i>.</p> <p>Required: No</p>
--cidr <i>ip_range</i>	<p>Specifies a CIDR/IP range for the security group to use for container instances in your cluster.</p> <p>Note This parameter is ignored if an existing security group is specified with the --security-group option.</p> <p>Type: CIDR/IP range</p> <p>Default: 0.0.0.0/0</p> <p>Required: No</p>
--port <i>port_number</i>	<p>Specifies a port to open on the security group to use for container instances in your cluster.</p> <p>Note This parameter is ignored if an existing security group is specified with the --security-group option.</p> <p>Type: Integer</p> <p>Default: 80</p> <p>Required: No</p>
--subnets <i>subnet_1</i> , <i>subnet_2</i>	<p>Specifies a comma-separated list of existing VPC subnet IDs in which to launch your container instances.</p> <p>Type: String</p> <p>Required: This option is required if you specify a VPC with the --vpc option.</p>

Name	Description
--vpc <i>vpc_id</i>	<p>Specifies the ID of an existing VPC in which to launch your container instances. If you specify a VPC ID, you must specify a list of existing subnets in that VPC with the --subnets option. If you do not specify a VPC ID, a new VPC is created with two subnets.</p> <p>Type: String</p> <p>Required: No</p>
--extra-user-data <i>string</i>	<p>Specifies additional user data for your container instance. Files can be shell scripts or cloud-init directives. They are packaged into a MIME multipart archive along with user data provided by the Amazon ECS CLI that directs instances to join your cluster. For more information, see Specifying User Data (p. 87).</p> <p>Type: String</p> <p>Required: No</p>
--instance-type <i>instance_type</i>	<p>Specifies the Amazon EC2 instance type for your container instances. If you specify an A1 instance type, for example <code>a1.medium</code>, and omit the --image-id parameter, the ECS CLI uses the the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI AMI ID for the container instance.</p> <p>Note This parameter is supported only with tasks that use the EC2 launch type.</p> <p>For more information on EC2 instance types, see Amazon EC2 Instances.</p> <p>Type: String</p> <p>Default: <code>t2.micro</code></p> <p>Required: No</p>

Name	Description
--image-id <i>ami_id</i>	<p>Specifies the Amazon EC2 AMI ID to use for your container instances.</p> <p>If you don't specify an AMI ID, the Amazon ECS CLI automatically retrieves the latest stable Amazon ECS-optimized Amazon Linux 2 AMI by querying the Systems Manager Parameter Store API during the cluster resource creation process. This requires the user account that you're using to have the required Systems Manager permissions. For more information, see Retrieving Amazon ECS-Optimized AMI metadata (p. 339).</p> <p>If you specify an A1 instance type for the --instance-type parameter and omit the --image-id parameter, the ECS CLI uses the the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI AMI ID for the container instance.</p> <p>Note This parameter is supported only with tasks that use the EC2 launch type.</p> <p>Type: String</p> <p>Default: The latest stable Amazon ECS-optimized AMI for the specified Region.</p> <p>Required: No</p>
--no-associate-public-ip-address	<p>Do not assign public IP addresses to new instances in this VPC. Unless this option is specified, new instances in this VPC receive an automatically assigned public IP address.</p> <p>Note This parameter is only supported with tasks that use the EC2 launch type.</p> <p>Required: No</p>
--force, -f	<p>Forces the recreation of any existing resources that match your current configuration. This option is useful for cleaning up stale resources from previous failed attempts.</p> <p>Required: No</p>
--tags <i>key1=value1, key2=value2</i>	<p>Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: <i>key1=value1, key2=value2, key3=value3</i>. For more information, see Tagging Resources (p. 87).</p> <p>Type: Key value pairs</p> <p>Required: No</p>

Name	Description
--instance-role, -f <i>instance-profile-name</i>	<p>Specifies a custom IAM role name for instances in your cluster. A new instance profile will be created and attached to this role.</p> <p>Note This parameter is only supported with tasks that use the EC2 launch type.</p> <p>This parameter is required if you do not specify the --capability-iam option. You cannot specify both options.</p> <p>Required: No</p>
--launch-type <i>launch_type</i>	<p>Specifies the launch type to use. Available options are FARGATE or EC2. For more information about launch types, see Amazon ECS launch types (p. 247).</p> <p>This overrides the default launch type stored in your cluster configuration.</p> <p>Type: String</p> <p>Required: No</p>
--verbose, --debug	<p>Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues.</p> <p>Required: No</p>
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--cluster-config <i>cluster_config_name</i>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the profile command.</p> <p>Type: String</p> <p>Required: No</p>
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>

Name	Description
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--empty, -e	<p>Specifies that an ECS cluster is created with no resources. If other flags are also specified that would create resources, they are ignored and a warning is displayed.</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Specifying User Data

When launching tasks that use the EC2 launch type, the ECS CLI always creates container instances that include the following user data:

```
#!/bin/bash
echo ECS_CLUSTER={ clusterName } >> /etc/ecs/ecs.config
```

This user data directs the container instance to join your ECS cluster. You can optionally include additional user data using the **--extra-user-data** flag. The flag can be specified multiple times. For example, extra user data can be shell scripts or cloud-init directives. For more information, see [Running Commands on Your Linux Instance at Launch](#) in the *Amazon EC2 User Guide for Linux Instances*.

The Amazon ECS CLI takes the user data and packs it into a MIME multipart archive, which can be used by cloud-init on the container instance. The Amazon ECS CLI allows existing MIME multipart archives to be passed in with **--extra-user-data**. The Amazon ECS CLI unpacks the existing archive, and then repack it into the final archive (preserving all header and content type information). The following is an example:

```
ecs-cli up \
--capability-iam \
--extra-user-data my-shellscript \
--extra-user-data my-cloud-boot-hook \
--extra-user-data my-mime-multipart-archive \
--launch-type EC2
```

Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

If you specify resource tags when using the **ecs-cli up** command, the Amazon ECS cluster as well as the following resources created by the AWS CloudFormation stack can be tagged:

- Container instances

Note

In order for your container instances to allow tags, you need to opt in to the new Amazon ECS resource ARN formats. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 325\)](#).

- VPC
- Subnets
- Internet gateway
- Route tables
- Security group
- Autoscaling group

Note

For the autoscaling group, the ECS CLI adds a Name tag whose value is the `ECS Instance - <CloudFormation stack name>`, which is propagated to your container instances. You can override this behavior by specifying your own Name tag.

Examples

Creating a Cluster for Tasks Using the EC2 Launch Type

This example brings up a cluster of four `c4.large` container instances and configures them to use the EC2 key pair called `id_rsa`.

```
ecs-cli up --keypair id_rsa --capability-iam --size 4 --instance-type c4.large --launch-type EC2
```

Output:

```
INFO[0001] Using recommended Amazon Linux AMI with ECS Agent 1.17.3 and Docker version 17.12.1-ce
INFO[0000] Created cluster cluster=ecs-cli-ec2-demo
INFO[0000] Waiting for your cluster resources to be created
INFO[0001] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0061] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0121] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0181] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
Cluster creation succeeded.
VPC created: vpc-abcd1234
Security Group created: sg-abcd1234
Subnets created: subnet-abcd1234
Subnets created: subnet-dcba4321
```

Creating a Cluster with Container Instances That Use the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI

This example brings up a cluster of one `a1.medium` container instances which will use the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI.

```
ecs-cli up --capability-iam --instance-type a1.medium --launch-type EC2 --region us-east-2
```

Output:

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.
INFO[0000] Using Arm ecs-optimized AMI because instance type was a1.medium
```

```
INFO[0001] Using recommended Amazon Linux 2 AMI with ECS Agent 1.25.3 and Docker version
18.06.1-ce
INFO[0000] Created cluster cluster=ecs-cli-ec2-demo
INFO[0000] Waiting for your cluster resources to be created
INFO[0001] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0061] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0121] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
INFO[0181] Cloudformation stack status stackStatus=CREATE_IN_PROGRESS
Cluster creation succeeded.
VPC created: vpc-abcd1234
Security Group created: sg-abcd1234
Subnets created: subnet-abcd1234
Subnets created: subnet-dcba4321
```

Creating a Cluster for Tasks Using the Fargate Launch Type

This example brings up a cluster for your Fargate tasks and creates a new VPC with two subnets.

```
ecs-cli up --launch-type FARGATE
```

Output:

```
INFO[0001] Created cluster cluster=ecs-cli-fargate-demo
region=us-west-2
INFO[0003] Waiting for your cluster resources to be created...
INFO[0003] Cloudformation stack status stackStatus="CREATE_IN_PROGRESS"
INFO[0066] Waiting for your cluster resources to be created...
INFO[0066] Cloudformation stack status stackStatus="CREATE_IN_PROGRESS"
VPC created: vpc-abcd1234
Subnets created: subnet-abcd1234
Subnets created: subnet-dcba4321
Cluster creation succeeded.
```

Creating an Empty Cluster

This example brings up an empty cluster named `ecs-cli-empty-demo` with no resources.

```
ecs-cli up --empty --cluster ecs-cli-empty-demo
```

Output:

```
INFO[0000] Created cluster cluster=ecs-cli-empty-demo
region=us-east-1
Cluster creation succeeded.
```

ecs-cli down

Deletes the AWS CloudFormation stack that was created by `ecs-cli up` and the associated resources.

Note

The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren't created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

The `ecs-cli down` command attempts to delete the cluster specified in `~/.ecs/config`. However, if there are any active services (even with a desired count of 0) or registered container instances in your cluster that were not created by `ecs-cli up`, the cluster is not deleted and the services and pre-existing

container instances remain active. This might happen, for example, if you used an existing ECS cluster with registered container instances, such as the default cluster.

If you have remaining services or container instances in your cluster that you would like to remove, you can follow the procedures in [Deleting a cluster \(p. 192\)](#) to delete your cluster.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli down [--force] [--cluster cluster_name] [--region region] [--help]
```

Options

Name	Description
--force, -f	Acknowledges that this command permanently deletes resources and bypasses the confirmation prompt. Required: No
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No

Name	Description
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example 1

This example deletes a cluster that contains resources.

```
ecs-cli down --cluster ecs-cli-fargate-demo --force
```

Output:

```
INFO[0001] Waiting for your cluster resources to be deleted
INFO[0001] Cloudformation stack status stackStatus=DELETE_IN_PROGRESS
INFO[0062] Cloudformation stack status stackStatus=DELETE_IN_PROGRESS
INFO[0123] Cloudformation stack status stackStatus=DELETE_IN_PROGRESS
INFO[0154] Deleted cluster
```

Example 2

This example deletes an empty cluster.

```
ecs-cli down --cluster ecs-cli-empty-demo --force
```

Output:

```
INFO[0002] No CloudFormation stack found for cluster 'ecs-cli-empty-demo'.
INFO[0003] Deleted cluster cluster=ecs-cli-empty-demo
```

ecs-cli scale

Modifies the number of container instances in your cluster. This command changes the desired and maximum instance count in the Auto Scaling group created by the **ecs-cli up** command. You can use this command to scale out (increase the number of instances) or scale in (decrease the number of instances) your cluster.

Note

The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren't created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli scale --capability-iam --size n [--cluster cluster_name] [--region region] [--help]
```

Options

Name	Description
--capability-iam	Acknowledges that this command may create IAM resources. Required: Yes
--size <i>n</i>	Specifies the number of instances to maintain in your cluster. Type: Integer Required: Yes
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example

This example scales the current cluster to two container instances.

```
ecs-cli scale --size 2 --capability-iam
```

Output:

```
INFO[0001] Waiting for your cluster resources to be updated
INFO[0001] Cloudformation stack status stackStatus=UPDATE_IN_PROGRESS
```

ecs-cli ps

Lists all running containers in your Amazon ECS cluster.

The IP address displayed by the Amazon ECS CLI depends heavily upon how you have configured your task and cluster:

- For tasks using the EC2 launch type without task networking, the IP address shown is the public IP address of the Amazon EC2 instance running your task, or the instance private IP address if it lacks a public IP address.
- For tasks using the EC2 launch type with task networking, the ECS CLI only shows a private IP address obtained from the network interfaces section of the Describe Task output for the task.
- For tasks using the Fargate launch type, the Amazon ECS CLI returns the public IP address assigned to the elastic network instance attached to the Fargate task. If the elastic network instance lacks a public IP address, then the Amazon ECS CLI falls back to the private IP address obtained from the network interfaces section of the Describe Task output.

Syntax

```
ecs-cli ps [--desired-status status] [--cluster cluster_name] [--region region] [--help]
```

Options

Name	Description
--desired-status <i>status</i>	The container desired status to filter the container list results with. Required: No Valid values: RUNNING STOPPED
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.

Name	Description
	Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example

This example shows the containers that are running in the cluster.

```
ecs-cli ps
```

Output:

Name	Health	State	Ports
afdf7f8a0-3813-4e1a-9d9e-ca7e9d1fcfb/wordpress	RUNNING	36.253.177.221:80->80/tcp	
compose3:7	HEALTHY		
dca67e02-68ca-4507-b194-a47239b5e7a9/wordpress	RUNNING	37.234.146.14:80->80/tcp	
healthcheck:3	UNKNOWN		
dca67e02-68ca-4507-b194-a47239b5e7a9/redis	RUNNING		
healthcheck:3	HEALTHY		
feb6e10e-3385-4c9b-a6cb-787cc8e90dda/sample-app	RUNNING	54.229.211.206:80->80/tcp	
tutorial-task-def:1	UNKNOWN		

ecs-cli push

Pushes an image to an Amazon ECR repository.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli push [--registry-id registry_id] [--tags key1=value1,key2=value2] [--region region] [--verbose] [--use-fips] ECR_REPOSITORY[:TAG] [--help]
```

Options

Name	Description
--registry-id <i>registry_id</i>	<p>Specifies the Amazon ECR registry ID to which to push the image. By default, images are pushed to the current AWS account.</p> <p>Type: String</p> <p>Required: No</p>
--tags <i>value</i>	<p>Specifies the metadata to apply to your Amazon ECR repository. Each tag consists of a key and an optional value. Tag keys can have a maximum character length of 128 characters, and tag values can have a maximum length of 256 characters. Tags use the following format: <i>key1=value1,key2=value2,key3=value3</i>.</p> <p>Type: Key value pairs</p> <p>Required: No</p>
--verbose, --debug	<p>Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues.</p> <p>Required: No</p>
--use-fips	<p>Routes calls to Amazon ECR through FIPS endpoints.</p> <p>Required: No</p>
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--cluster-config <i>cluster_config_name</i>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</p>

Name	Description
	Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Using FIPS Endpoints

The Amazon ECS CLI supports using FIPS endpoints for calls to Amazon ECR. To ensure that you're accessing Amazon ECR using FIPS endpoints, use the `--use-fips` flag on the push, pull, or images command. FIPS endpoints are currently available in us-west-1, us-west-2, us-east-1, us-east-2, and AWS GovCloud (US). For more information, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

Examples

Example 1

This example pushes a local image called `ubuntu` to an Amazon ECR repository with the same name.

```
ecs-cli push ubuntu
```

Output:

```
INFO[0000] Getting AWS account ID...
INFO[0000] Tagging image
repository="aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu" source-image=ubuntu
tag=
INFO[0000] Image tagged
INFO[0001] Creating repository
INFO[0001] Repository created
repository=ubuntu
INFO[0001] Pushing image
repository="aws_account_id.dkr.ecr.us-east-1.amazonaws.com/ubuntu" tag=
INFO[0079] Image pushed
```

ecs-cli pull

Pull an image from an Amazon ECR repository.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli pull [--registry-id registry_id] [--region region] [--verbose] [--use-fips] ECR_REPOSITORY[:TAG|@DIGEST] [--help]
```

Options

Name	Description
--registry-id <i>registry_id</i>	Specifies the Amazon ECR registry ID from which to pull the image. By default, images are pulled from the current AWS account. Required: No
--verbose, --debug	Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No
--use-fips	Routes calls to Amazon ECR through FIPS endpoints. Required: No
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No

Name	Description
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Using FIPS Endpoints

The Amazon ECS CLI supports using FIPS endpoints for calls to Amazon ECR. To ensure that you're accessing Amazon ECR using FIPS endpoints, use the `--use-fips` flag on the `push`, `pull`, or `images` command. FIPS endpoints are currently available in us-west-1, us-west-2, us-east-1, us-east-2, and AWS GovCloud (US). For more information, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

Examples

Example 1

This example pulls a local image called `amazonlinux` from an Amazon ECR repository with the same name.

```
ecs-cli pull amazonlinux
```

Output:

```
INFO[0000] Getting AWS account ID...
INFO[0000] Pulling image
repository="aws_account_id.dkr.ecr.us-east-1.amazonaws.com/amazonlinux" tag=
INFO[0129] Image pulled
```

ecs-cli images

List images in an Amazon ECR registry or repository.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli images [--registry-id registry_id] [--tagged|--untagged] [--region region] [--verbose] [--use-fips] [ECR_REPOSITORY] [--help]
```

Options

Name	Description
--registry-id <i>registry_id</i>	Specifies the Amazon ECR registry with which to list images. By default, images are listed for the current AWS account.

Name	Description
	Required: No
--tagged	Filters the result to show only tagged images. Required: No
--untagged	Filters the result to show only untagged images. Required: No
--verbose, --debug	Turn on debug logging. This provides a more verbose command output to aid in diagnosing issues. Required: No
--use-fips	Routes calls to Amazon ECR through FIPS endpoints. Required: No
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Using FIPS Endpoints

The Amazon ECS CLI supports using FIPS endpoints for calls to Amazon ECR. To ensure you are accessing Amazon ECR using FIPS endpoints, use the `--use-fips` flag on the `push`, `pull`, or `images` command. For more information, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

Examples

Example 1

This example lists all of the images in an Amazon ECR registry.

```
ecs-cli images
```

Output:

REPOSITORY NAME	TAG	PUSHED AT	IMAGE DIGEST	SIZE
rkt	latest			
sha256:404758ad8af94347fc8582fc8e30b6284f2b0751de29b2e755da212f80232fac		3 months ago		203 MB
foobuntu	latest			
sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8		4 days ago		51.7 MB
ubuntu	xenial			
sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8		4 days ago		51.7 MB
ubuntu	latest			
sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8		4 days ago		51.7 MB
ubuntu	<none>			
sha256:512e30a26d9fa3648dbccb9e78e9bab636e6022e2d80bd73c99177b21a0d3982		19 minutes ago		268 MB
ubuntu	trusty			
sha256:bd6d24e8fa3f5822146b2c94247976b87e6564195c3c180b67833e6ea699f7c2		18 minutes ago		67.2 MB
ubuntu	precise			
sha256:b38267a51fb4460699bc2bcd53d42fec697bb4e4f9a819df3e762cec393b2a		17 minutes ago		40.1 MB
amazon-ecs-sample	latest			
sha256:bf04071a8edecc309f4d109ae36f24a5c272a115b6f7e636f77940059024d71c		2 weeks ago		105 MB
golang	latest			
sha256:137b22efee2df470b0cd28ebfc1ae583be0baf09334a5a882096193577d983ab		4 days ago		266 MB
amazonlinux	latest			
sha256:a59d563b5139deee8cb108fb97bf3e9021b8cceaa6dec8ff49733230cb2f0eca		4 days ago		98.8 MB
awsbatch/fetch_and_run	latest			
sha256:543800007416d0ccff4f63643bb18eff4b874ea772128efcdc231ff456a37fc		6 weeks ago		116 MB

Example 2

This example lists all of the images in a specific Amazon ECR repository.

```
ecs-cli images ubuntu
```

Output:

REPOSITORY NAME	TAG	PUSHED AT	IMAGE DIGEST	SIZE
ubuntu	xenial		sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8	4 days ago 51.7 MB
ubuntu	latest		sha256:6b079ae764a6affcb632231349d4a5e1b084bece8c46883c099863ee2aeb5cf8	4 days ago 51.7 MB
ubuntu	<none>		sha256:512e30a26d9fa3648dbccb9e78e9bab636e6022e2d80bd73c99177b21a0d3982	20 minutes ago 268 MB
ubuntu	trusty		sha256:bd6d24e8fa3f5822146b2c94247976b87e6564195c3c180b67833e6ea699f7c2	19 minutes ago 67.2 MB
ubuntu	precise		sha256:b38267a51fb4460699bc2bcd53d42fec697bb4e4f9a819df3e762cec393b2a	18 minutes ago 40.1 MB

Example 3

This example lists all of the untagged images in an Amazon ECR registry.

```
ecs-cli images --untagged
```

Output:

REPOSITORY NAME	TAG	PUSHED AT	IMAGE DIGEST	SIZE
ubuntu	<none>		sha256:512e30a26d9fa3648dbccb9e78e9bab636e6022e2d80bd73c99177b21a0d3982	24 minutes ago 268 MB

ecs-cli license

Prints the LICENSE files for the Amazon ECS CLI and its dependencies.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli license [--help]
```

Options

Name	Description
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example

This example prints the license files.

```
ecs-cli license
```

Output:

```
Copyright 2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"). You may not use this file
except in compliance with the
License. A copy of the License is located at

http://aws.amazon.com/apache2.0/

or in the "license" file accompanying this file. This file is distributed on an "AS IS"
BASIS, WITHOUT WARRANTIES OR
CONDITIONS OF ANY KIND, either express or implied. See the License for the specific
language governing permissions
and limitations under the License.

...
```

ecs-cli compose

Manage Amazon ECS tasks with **docker-compose**-style commands on an ECS cluster. For more information on how Docker Compose file syntax works with the Amazon ECS CLI, see [Using Docker Compose File Syntax \(p. 153\)](#).

Note

To create Amazon ECS services with the Amazon ECS CLI, see [ecs-cli compose service \(p. 114\)](#).

The **ecs-cli compose** command uses a project name with the task definitions and services it creates. When the CLI creates a task definition from a Compose file, the task definition is called *project-name*. When the CLI creates a service from a Compose file, the service is called *service-project-name*. By default, the project name is the name of the directory that contains your Docker Compose file. However, you can also specify your own project name with the --project-name option.

Note

The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren't created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli compose [--verbose] [--file compose_file] [--project-name project_name] [--task-role-arn
task_role_arn] [--ecs-params ecs_params_file] [--registry-creds value] [--region region] [--cluster-config cluster_config_name] [-ecs-profile ecs_profile] [-aws-profile aws_profile] [--cluster cluster_name] [-help] [subcommand] [arguments] [-help]
```

Options

Name	Description
--verbose, --debug	Increases the verbosity of command output to aid in diagnostics. Required: No

Name	Description
--file, -f <i>compose_file</i>	<p>Specifies the Docker Compose file to use. At this time, the latest version of the Amazon ECS CLI only supports the major versions of Docker Compose file syntax versions 1, 2, and 3. The version specified in the compose file must be the string "1", "1.0", "2", "2.0", "3", or "3.0". Docker Compose minor versions are not supported. If the <code>COMPOSE_FILE</code> environment variable is set when <code>ecs-cli compose</code> is run, the Docker Compose file is set to the value of that environment variable.</p> <p>Type: String</p> <p>Default: <code>./docker-compose.yml</code></p> <p>Required: No</p>
--project-name, -p <i>project_name</i>	<p>Specifies the project name to use. If the <code>COMPOSE_PROJECT_NAME</code> environment variable is set when <code>ecs-cli compose</code> is run, the project name is set to the value of that environment variable.</p> <p>Type: String</p> <p>Default: The current directory name.</p> <p>Required: No</p>
--task-role-arn <i>role_value</i>	<p>Specifies the short name or full Amazon Resource Name (ARN) of the IAM role that containers in this task can assume. All containers in this task are granted the permissions that are specified in this role.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-params <i>ecs_params_file</i>	<p>Specifies the ECS parameters that aren't native to Docker Compose files. For more information, see Using Amazon ECS Parameters (p. 155).</p> <p>Default: <code>./ecs-params.yml</code></p> <p>Required: No</p>
--registry-creds <i>value</i>	<p>Specifies the Amazon ECS registry credentials file to use. Defaults to the latest output file from the <code>ecs-cli registry-creds up</code> command, if one exists. For more information, see ecs-cli registry-creds (p. 140).</p> <p>Default: <code>./ecs-registry-creds_[TIMESTAMP].yml</code></p> <p>Required: No</p>

Name	Description
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Available Subcommands

The **ecs-cli compose** command supports the following subcommands. Each of these subcommands has their own flags associated with them, which can be displayed using the `--help` flag.

create

Creates an Amazon ECS task definition from your Compose file. For more information, see [ecs-cli compose create \(p. 105\)](#).

ps, list

Lists all the containers in your cluster that were started by the Compose project.

run [*containerName*] ["*command* ..."] ...

Starts all containers overriding commands with the supplied one-off commands for the containers.

scale *n*

Scales the number of running tasks to the specified count.

start

Starts a single task from the task definition created from your Compose file. For more information, see [ecs-cli compose start \(p. 108\)](#).

stop, down

Stops all the running tasks created by the Compose project.

up

Creates an ECS task definition from your Compose file (if it doesn't already exist) and runs one instance of that task on your cluster (a combination of **create** and **start**). For more information, see [ecs-cli compose up \(p. 111\)](#).

service [*subcommand*]

Creates an ECS service from your Compose file. For more information, see [ecs-cli compose service \(p. 114\)](#).

help

Shows the help text for the specified command.

[ecs-cli compose create](#)

Creates an Amazon ECS task definition from your Compose file.

Important

We don't recommend using plaintext environment variables for sensitive information, such as credential data.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli compose create [--region region] [--cluster-config cluster_config_name] [--ecs-profile ecs_profile] [--aws-profile aws_profile] [--cluster cluster_name] [--launch-type launch_type] [--create-log-groups] [--tags key1=value1, key2=value2] [--help]
```

Options

Name	Description
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String

Name	Description
	Required: No
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</p> <p>Type: String</p> <p>Required: No</p>
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--launch-type <i>launch_type</i>	<p>Specifies the launch type to use. Available options are FARGATE or EC2. For more information about launch types, see Amazon ECS launch types (p. 247).</p> <p>This overrides the default launch type stored in your cluster configuration.</p> <p>Type: String</p> <p>Required: No</p>
--create-log-groups	<p>Creates the CloudWatch log groups specified in your Compose files.</p> <p>Required: No</p>
--tags <i>key1=value1, key2=value2</i>	<p>Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: <code>key1=value1, key2=value2, key3=value3</code>. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the <code>--disable-ecs-managed-tags</code> flag. For more information, see Tagging Resources (p. 130).</p> <p>Type: Key value pairs</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

When using the `ecs-cli compose create` command, using the `--tags` flag enables you to add metadata tags to the task definition.

Examples

Register a Task Definition

This example creates a task definition with the project name `hello-world` from the `hello-world.yml` Compose file.

For information about the `compose` options, see [the section called "Options" \(p. 102\)](#).

```
ecs-cli compose --project-name hello-world --file hello-world.yml create --launch-type EC2
```

Output:

INFO[0000] Using ECS task definition world:5	TaskDefinition=ecscompose-hello-
---	----------------------------------

Register a Task Definition Using the EC2 Launch Type Without Task Networking

This example creates a task definition with the project name `hello-world` from the `hello-world.yml` Compose file. Additional ECS parameters specified for the container size parameters.

Example Docker Compose file, named `hello-world.yml`:

```
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: /ecs/cli/tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named `ecs-params.yml`:

```
version: 1
task_definition:
  services:
    nginx:
      cpu_shares: 256
      mem_limit: 0.5GB
      mem_reservation: 0.5GB
```

```
ecs-cli compose --project-name hello-world --file hello-world.yml --ecs-params ecs-params.yml --region us-east-1 create --launch-type EC2
```

Output:

```
INFO[0000] Using ECS task definition          TaskDefinition=ecscompose-hello-
world:5
```

Register a Task Definition Using the Fargate Launch Type

This example creates a task definition with the project name hello-world from the hello-world.yml Compose file. Additional ECS parameters are specified for task networking configuration for the Fargate launch type. Then one instance of the task is run.

Example Docker Compose file, named hello-world.yml:

```
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named ecs-params.yml:

```
version: 1
task_definition:
  task_execution_role: ecsTaskExecutionRole
  ecs_network_mode: awsvpc
  task_size:
    mem_limit: 0.5GB
    cpu_limit: 256
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - subnet-abcd1234
        - subnet-dbca4321
      security_groups:
        - sg-abcd1234
  assign_public_ip: ENABLED
```

Command:

```
ecs-cli compose --project-name hello-world --file hello-world.yml --ecs-params ecs-
params.yml --region us-east-1 create --launch-type FARGATE
```

Output:

```
INFO[0000] Using ECS task definition          TaskDefinition=ecscompose-hello-
world:5
```

ecs-cli compose start

Starts a single Amazon ECS task from the task definition created from your Compose file.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli compose start [--region region] [--cluster-config cluster_config_name] [--ecs-profile ecs_profile] [--aws-profile aws_profile] [--cluster cluster_name] [--launch-type launch_type] [--create-log-groups] [--help]
```

Options

Name	Description
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--cluster-config <i>cluster_config_name</i>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</p> <p>Type: String</p> <p>Required: No</p>
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--launch-type <i>launch_type</i>	<p>Specifies the launch type to use. Available options are FARGATE or EC2. For more information about launch types, see Amazon ECS launch types (p. 247).</p> <p>This overrides the default launch type stored in your cluster configuration.</p> <p>Type: String</p>

Name	Description
	Required: No
--create-log-groups	Creates the CloudWatch log groups specified in your Compose files. Required: No
--help, -h	Shows the help text for the specified command. Required: No

Examples

Run a Task

This example creates a task definition from the `hello-world.yml` Compose file. Additional ECS parameters are specified for task networking configuration for the Fargate launch type. Then a single task is run using that task definition.

Example Docker Compose file, named `hello-world.yml`:

```
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named `ecs-params.yml`:

```
version: 1
task_definition:
  task_execution_role: ecsTaskExecutionRole
  ecs_network_mode: awsvpc
  task_size:
    mem_limit: 0.5GB
    cpu_limit: 256
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - subnet-abcd1234
        - subnet-dbca4321
      security_groups:
        - sg-abcd1234
  assign_public_ip: ENABLED
```

Command:

```
ecs-cli compose --file hello-world.yml --ecs-params ecs-params.yml start --launch-type FARGATE --create-log-groups
```

Output:

```
INFO[0000] Using ECS task definition           TaskDefinition=ecscompose-hello-
world:5
```

ecs-cli compose up

If an Amazon ECS task definition doesn't already exist, creates one from your Compose file and runs one instance of that task on your cluster.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli compose up [--region region] [--cluster-config cluster_config_name] [--ecs-profile ecs_profile] [--aws-profile aws_profile] [--cluster cluster_name] [--launch-type launch_type] [--create-log-groups] [--force-update] [--tags key1=value1, key2=value2] [--disable-ecs-managed-tags] [--help]
```

Options

Name	Description
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.

Name	Description
	Type: String Required: No
--launch-type <i>launch_type</i>	Specifies the launch type to use. Available options are FARGATE or EC2. For more information about launch types, see Amazon ECS launch types (p. 247) . This overrides the default launch type stored in your cluster configuration. Type: String Required: No
--create-log-groups	Creates the CloudWatch log groups specified in your Compose files. Required: No
--force-update	Forces the relaunching of the tasks. Required: No
--tags <i>key1=value1, key2=value2</i>	Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: key1=value1, key2=value2, key3=value3. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the --disable-ecs-managed-tags flag. For more information, see Tagging Resources (p. 130) . Type: Key value pairs Required: No
--disable-ecs-managed-tags	Disable the Amazon ECS managed tags. For more information, see Tagging your resources for billing (p. 606) . Required: No
--help, -h	Shows the help text for the specified command. Required: No

Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

When using the `ecs-cli compose up` command, using the `--tags` flag enables you to add metadata tags to the task definition and tasks. Amazon ECS managed tags are enabled by default unless specifically disabled using the `--disable-ecs-managed-tags` flag. For more information, see [Tagging your resources for billing \(p. 606\)](#).

Examples

Register a Task Definition Using the AWS Fargate Launch Type with Task Networking

This example creates a task definition with the project name `hello-world` from the `hello-world.yml` Compose file. Additional ECS parameters are specified for task and network configuration for the Fargate launch type. Then one instance of the task is run using the Fargate launch type.

Example Docker Compose file, named `hello-world.yml`:

```
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: us-east-1
        awslogs-stream-prefix: nginx
```

Example ECS parameters file, named `ecs-params.yml`:

```
version: 1
task_definition:
  ecs_network_mode: awsvpc
  task_execution_role: ecsTaskExecutionRole
  task_size:
    cpu_limit: 512
    mem_limit: 2GB
  services:
    nginx:
      essential: true
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - subnet-abcd1234
        - subnet-dcba4321
      security_groups:
        - sg-abcd1234
        - sg-dcba4321
      assign_public_ip: ENABLED
```

Command:

```
ecs-cli compose --project-name hello-world --file hello-world.yml --ecs-params ecs-params.yml up --launch-type FARGATE
```

Output:

```
INFO[0000] Using ECS task definition                               TaskDefinition=ecscompose-hello-world:5
```

ecs-cli compose service

Manage Amazon ECS services with **docker-compose**-style commands on an ECS cluster. For more information on how Docker compose file syntax works with the ECS CLI, see [Using Docker Compose File Syntax \(p. 153\)](#).

Note

To run tasks with the Amazon ECS CLI instead of creating services, see [ecs-cli compose \(p. 102\)](#).

The **ecs-cli compose service** command uses a project name with the task definitions and services that it creates. When the Amazon ECS CLI creates a task definition and service from a compose file, the task definition and service are called *project-name*. By default, the project name is the name of the directory that contains your Docker compose file. However, you can also specify your own project name with the `--project-name` option.

Note

The Amazon ECS CLI can only manage tasks, services, and container instances that were created with the Amazon ECS CLI. To manage tasks, services, and container instances that weren't created by the Amazon ECS CLI, use the AWS Command Line Interface or the AWS Management Console.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli compose [--verbose] [--file compose_file] [--project-name project_name]
[--task-role-arn task_role_arn] [--ecs-params ecs_params_file] [--registry-creds value] [--region region] [--cluster-config cluster_config_name] [--ecs-profile ecs_profile] [--aws-profile aws_profile] [--cluster cluster_name] [--help] service [subcommand] [arguments] [--help]
```

Options

Name	Description
<code>--verbose</code> , <code>--debug</code>	Increases the verbosity of command output to aid in diagnostics. Required: No
<code>--file</code> , <code>-f <i>compose_file</i></code>	Specifies the Docker Compose file to use. At this time, the latest version of the Amazon ECS CLI only supports the major versions of Docker Compose file syntax versions 1, 2, and 3. The version specified in the compose file must be the string "1", "1.0", "2", "2.0", "3", or "3.0". Docker Compose minor versions are not supported. If the <code>COMPOSE_FILE</code> environment variable is set when ecs-cli compose is run, the Docker Compose file is set to the value of that environment variable. Type: String Default: <code>./docker-compose.yml</code> Required: No

Name	Description
--project-name, -p <i>project_name</i>	<p>Specifies the project name to use. If the <code>COMPOSE_PROJECT_NAME</code> environment variable is set when <code>ecs-cli compose</code> is run, the project name is set to the value of that environment variable.</p> <p>Type: String</p> <p>Default: The current directory name.</p> <p>Required: No</p>
--task-role-arn <i>role_value</i>	<p>Specifies the short name or full Amazon Resource Name (ARN) of the IAM role that containers in this task can assume. All containers in this task are granted the permissions that are specified in this role.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-params <i>ecs_params_file</i>	<p>Specifies the ECS parameters that aren't native to Docker Compose files. For more information, see Using Amazon ECS Parameters (p. 155).</p> <p>Default: <code>./ecs-params.yml</code></p> <p>Required: No</p>
--registry-creds <i>value</i>	<p>Specifies the Amazon ECS registry credentials file to use. Defaults to the latest output file from the <code>ecs-cli registry-creds up</code> command, if one exists. For more information, see ecs-cli registry-creds (p. 140).</p> <p>Default: <code>./ecs-registry-creds_[TIMESTAMP].yml</code></p> <p>Required: No</p>
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</p> <p>Type: String</p> <p>Required: No</p>
--cluster-config <i>cluster_config_name</i>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.</p> <p>Type: String</p> <p>Required: No</p>

Name	Description
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</p> <p>Type: String</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Available Subcommands

The `ecs-cli compose service` command supports the following subcommands. Each of these subcommands has their own flags associated with them, which can be displayed using the `--help` flag.

`create`

Creates an Amazon ECS service from your compose file. The service is created with a desired count of 0, so no containers are started by this command. For more information, see [ecs-cli compose service create \(p. 117\)](#).

`start`

Starts one copy of each of the containers on the created Amazon ECS service. This command updates the desired count of the service to 1. For more information, see [ecs-cli compose service start \(p. 122\)](#).

`up`

Creates an Amazon ECS service from your compose file (if it does not already exist) and runs one instance of that task on your cluster (a combination of `create` and `start`). This command updates the desired count of the service to 1. For more information, see [ecs-cli compose service up \(p. 124\)](#).

`ps, list`

Lists all the containers in your cluster that belong to the service created with the compose project. For more information, see [ecs-cli compose service ps, list \(p. 131\)](#).

`scale`

Scales the desired count of the service to the specified count. For more information, see [ecs-cli compose service scale \(p. 132\)](#).

`stop`

Stops the running tasks that belong to the service created with the compose project. This command updates the desired count of the service to 0. For more information, see [ecs-cli compose service stop \(p. 134\)](#).

`rm, delete, down`

Updates the desired count of the service to 0 and then deletes the service. For more information, see [ecs-cli compose service rm, delete, down \(p. 135\)](#).

ecs-cli compose service create

Creates an Amazon ECS service from your compose file. The service is created with a desired count of 0, so no containers are started by this command.

Syntax

```
ecs-cli compose service create [--deployment-max-percent n] [--deployment-min-  
healthy-percent n] [-load-balancer-name value] [--target-group-arn value] [--  
target-groups "targetGroupArn=arn:elasticloadbalancing:region:aws_account_id:  
targetgroup/target_group_name_1,containerName=container_name,containerPort=container_port"]  
[--container-name value] [--container-port value] [--role value] [--launch-type launch_type]  
[--health-check-grace-period integer] [--create-log-groups] [--enable-service-discovery] [--  
vpc value] [--private-dns-namespace value] [--private-dns-namespace-id value] [--public-dns-  
namespace value] [--public-dns-namespace-id value] [--sd-container-name value] [--sd-container-  
port value] [--dns-ttl value] [--dns-type value] [--healthcheck-custom-config-failure-threshold  
value] [--scheduling-strategy value] [--tags key1=value1, key2=value2] [--disable-ecs-managed-  
tags] [--help]
```

Options

Name	Description
--deployment-max-percent	<p>Specifies the upper limit (as a percentage of the service's <code>desiredCount</code>) of the number of running tasks that can be running in a service during a deployment. For more information, see maximumPercent (p. 537).</p> <p>Default value: 200</p> <p>Required: No</p>
--deployment-min-healthy-percent	<p>Specifies the lower limit (as a percentage of the service's <code>desiredCount</code>) of the number of running tasks that must remain running and healthy in a service during a deployment. For more information, see minimumHealthyPercent (p. 538).</p> <p>Default value: 100</p> <p>Required: No</p>
--target-group-arn	<p>Specifies the full Amazon Resource Name (ARN) of a previously configured Elastic Load Balancing target group to associate with your service.</p> <p>Deprecated, use <code>target-groups</code>.</p> <p>Required: No</p>
--target-groups	<p>Specifies one or more target groups to be registered with the service. Specify the Amazon Resource Name (ARN) of the target group, the container name and the container port for each group to register. A single <code>--target-groups</code> flag specifies only one target group. Add additional <code>--target-groups</code> flags to add additional target groups.</p> <p>You use a target group to configure where a load balancer direct traffic to, for example Amazon EC2 instances. When you create</p>

Name	Description
	<p>a load balancer, you create one or more listeners and configure listener rules to direct the traffic to one target group.</p> <p>Type: Strings</p> <p>For more information see Registering multiple target groups with a service (p. 589).</p> <p>Required: No</p>
--container-name	<p>Specifies the container name (as it appears in a container definition). This parameter is required if a load balancer or target group is specified.</p> <p>Deprecated, use <code>target-groups</code>.</p> <p>Required: No, unless a load balancer or target group is specified.</p>
--container-port	<p>Specifies the port on the container to associate with the load balancer. This port must correspond to a <code>containerPort</code> in the service's task definition. This parameter is required if a load balancer or target group is specified.</p> <p>Deprecated, use <code>target-groups</code>.</p> <p>Required: No, unless a load balancer or target group is specified.</p>
--load-balancer-name	<p>Specifies the name of a previously configured Elastic Load Balancing load balancer to associate with your service.</p> <p>Deprecated, use <code>target-groups</code>.</p> <p>Required: No</p>
--role	<p>Specifies the name or full Amazon Resource Name (ARN) of the IAM role that allows Amazon ECS to make calls to your load balancer or target group on your behalf. This parameter is required if you're using a load balancer or target group with your service. If you specify the role parameter, you must also specify a load balancer name or target group ARN, along with a container name and container port.</p> <p>Required: No, unless a load balancer or target group is specified.</p>
--health-check-grace-period	<p>Specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started.</p> <p>Required: No</p>
--create-log-groups	<p>Creates the CloudWatch log groups specified in your Compose files.</p> <p>Required: No</p>
--enable-service-discovery	<p>Specifies whether to enable service discovery for this service.</p> <p>Required: No</p>

Name	Description
--vpc	<p>Specifies the VPC that will be attached to the private DNS namespace for service discovery. This parameter is required if --private-dns-namespace is specified.</p> <p>Required: No</p>
--private-dns-namespace	<p>Specifies the name of the private DNS namespace to use with service discovery. The Amazon ECS CLI automatically creates the namespace if it doesn't exist. For example, if the namespace is <code>corp</code>, a service named <code>foo</code> is reachable via DNS at <code>foo.corp</code>. If you use this parameter, you must also specify a VPC using the --vpc parameter.</p> <p>Required: No</p>
--private-dns-namespace-id	<p>Specifies the ID of an existing private DNS namespace to use with service discovery. If you use this parameter, you can't specify either --private-dns-namespace or --vpc.</p> <p>Required: No</p>
--public-dns-namespace	<p>Specifies the name of the public DNS namespace to use with service discovery. For example, if the namespace is <code>corp</code>, a service named <code>foo</code> is reachable via DNS at <code>foo.corp</code>.</p> <p>Required: No</p>
--public-dns-namespace-id	<p>Specifies the ID of an existing public DNS namespace to use with service discovery. If you use this parameter, you can't specify a --public-dns-namespace.</p> <p>Required: No</p>
--sd-container-name	<p>Specifies the name of the container, which is referred to as a service in your Docker Compose file. For more information, see Service configuration reference. This parameter is required if you're using SRV records.</p> <p>Required: No, unless SRV DNS records are being used.</p>
--sd-container-port	<p>Specifies the port on the container that will be used for service discovery. This parameter is required if you're using SRV records.</p> <p>Required: No, unless SRV DNS records are being used.</p>
--dns-ttl	<p>Specifies the amount of time, in seconds, that you want DNS resolvers to cache the settings for the DNS records used for service discovery.</p> <p>Default value: 60</p> <p>Required: No</p>

Name	Description
--dns-type	<p>Specifies the type of DNS record used for service discovery. Accepted values are <code>A</code> or <code>SRV</code>. If your task uses either the <code>bridge</code> or <code>host</code> network modes, <code>SRV</code> records are required. If your task uses the <code>awsvpc</code> network mode, <code>A</code> records are the default.</p> <p>Required: No</p>
--healthcheck-custom-config-failure-threshold	<p>Specifies the number of 30-second intervals that you want the service discovery service to wait after receiving an <code>UpdateInstanceCustomHealthStatus</code> request before it changes the health status.</p> <p>Default value: 1</p> <p>Required: No</p>
--scheduling-strategy <i>value</i>	<p>Specifies the scheduling strategy to use for the service.</p> <p>There are two service scheduler strategies available:</p> <ul style="list-style-type: none"> • 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. 533). • DAEMON—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon (p. 532). <p>Note Fargate tasks do not support the <code>DAEMON</code> scheduling strategy.</p> <p>For more information, see Service scheduler concepts (p. 531).</p> <p>Type: String</p> <p>Valid values: <code>REPLICA</code> <code>DAEMON</code></p> <p>Default value: <code>REPLICA</code></p> <p>Required: No</p>

Name	Description
--tags <i>key1=value1, key2=value2</i>	<p>Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: key1=value1, key2=value2, key3=value3. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the --disable-ecs-managed-tags flag. For more information, see Tagging Resources (p. 130).</p> <p>Type: Key value pairs</p> <p>Required: No</p>
--disable-ecs-managed-tags	<p>Disable the Amazon ECS managed tags. For more information, see Tagging your resources for billing (p. 606).</p> <p>Required: No</p>
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--cluster-config <i>cluster_config_name</i>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</p> <p>Type: String</p> <p>Required: No</p>
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Using a Load Balancer

You can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service. For more information, see [Service load balancing \(p. 574\)](#). After you create a service, you can't change the load balancer name or target group ARN, container name, and container port specified in the service definition.

Note

You must create your load balancer resources before you can configure a service to use them.

Your load balancer resources should reside in the same VPC as your container instances, and they should be configured to use the same subnets. You must also add a security group rule to your container instance security group that allows inbound traffic from your load balancer. For more information, see [Creating a load balancer \(p. 579\)](#).

- To configure your service to use an existing Elastic Load Balancing Classic Load Balancer, you must specify the load balancer name, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance is registered with the load balancer specified here.
- To configure your service to use an existing Elastic Load Balancing Application Load Balancer, you must specify the load balancer target group ARN, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance and port combination is registered as a target in the target group specified here.

The `--health-check-grace-period` option specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started. This is valid only if your service is configured to use a load balancer. If your tasks take a while to start and respond to Elastic Load Balancing health checks, you can specify a health check grace period of up to 1,800 seconds during which the Amazon ECS service scheduler ignores the Elastic Load Balancing health check status. This grace period can prevent the Amazon ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

Using Service Discovery

Your Amazon ECS service can optionally be configured to use Amazon ECS Service Discovery. Service discovery uses Amazon Route 53 auto naming API actions to manage DNS entries for your service's tasks, making them discoverable within your VPC. For more information, see [Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI \(p. 70\)](#).

Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

When using the `ecs-cli compose service create` command, using the `--tags` flag allows you to add metadata tags to the task definition and service. The tags are added to the service and task definition when the resources are created. The tags are propagated from your task definition to tasks created by the service. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the `--disable-ecs-managed-tags` flag. For more information, see [Tagging your resources for billing \(p. 606\)](#).

ecs-cli compose service start

Starts one copy of each of the containers on the created Amazon ECS service. This command updates the desired count of the service to 1.

Syntax

```
ecs-cli compose service start [--create-log-groups] [--force-deployment] [--help]
```

Options

Name	Description
--timeout <i>value</i>	<p>Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes).</p> <p>Default value: 5</p> <p>Required: No</p>
--create-log-groups	<p>Creates the CloudWatch log groups specified in your Compose files.</p> <p>Required: No</p>
--force-deployment	<p>Forces a new deployment of the service.</p> <p>Required: No</p>
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--cluster-config <i>cluster_config_name</i>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</p> <p>Type: String</p> <p>Required: No</p>
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p>

Name	Description
	Required: No
--help, -h	Shows the help text for the specified command. Required: No

ecs-cli compose service up

Creates an Amazon ECS service from your compose file (if it does not already exist) and runs one instance of that task on your cluster (a combination of the **create** and **start** commands). This command updates the desired count of the service to 1.

Syntax

```
ecs-cli compose service up [--deployment-max-percent n] [--deployment-min-healthy-percent n] [--load-balancer-name value] [--target-group-arn value] [--target-groups "targetGroupArn=arn:elasticloadbalancing:region:aws_account_id:targetgroup/target_group_name_1,containerName=container_name,containerPort=container_port"] [--container-name value] [--container-port value] [--role value] [--health-check-grace-period integer] [--timeout value] [--launch-type launch_type] [--create-log-groups] [--force-deployment] [--enable-service-discovery] [--vpc value] [--private-dns-namespace value] [--private-dns-namespace-id value] [--public-dns-namespace value] [--public-dns-namespace-id value] [--sd-container-name value] [--sd-container-port value] [--dns-ttl value] [--dns-type value] [--healthcheck-custom-config-failure-threshold value] [--update-service-discovery] [--scheduling-strategy value] [--tags key1=value1, key2=value2] [--disable-ecs-managed-tags] [--help]
```

Options

Name	Description
--deployment-max-percent	Specifies the upper limit (as a percentage of the service's <code>desiredCount</code>) of the number of running tasks that can be running in a service during a deployment. For more information, see maximumPercent (p. 537) . Default value: 200 Required: No
--deployment-min-healthy-percent	Specifies the lower limit (as a percentage of the service's <code>desiredCount</code>) of the number of running tasks that must remain running and healthy in a service during a deployment. For more information, see minimumHealthyPercent (p. 538) . Default value: 100 Required: No
--target-group-arn	Specifies the full Amazon Resource Name (ARN) of a previously configured Elastic Load Balancing target group to associate with your service. Deprecated , use <code>target-groups</code> . Required: No

Name	Description
--target-groups	<p>Specifies one or more target groups to be registered with the service. Specify the Amazon Resource Name (ARN) of the target group, the container name and the container port for each group to register. A single --target-groups flag specifies only one target group. Add additional --target-groups flags to add additional target groups.</p> <p>You use a target group to configure where a load balancer directs traffic to, for example Amazon EC2 instances. When you create a load balancer, you create one or more listeners and configure listener rules to direct the traffic to one target group.</p> <p>Type: Strings</p> <p>For more information see Registering multiple target groups with a service (p. 589).</p> <p>Required: No</p>
--container-name	<p>Specifies the container name (as it appears in a container definition). This parameter is required if a load balancer or target group is specified.</p> <p>Deprecated, use target-groups.</p> <p>Required: No, unless a load balancer or target group is specified.</p>
--container-port	<p>Specifies the port on the container to associate with the load balancer. This port must correspond to a containerPort in the service's task definition. This parameter is required if a load balancer or target group is specified.</p> <p>Deprecated, use target-groups.</p> <p>Required: No, unless a load balancer or target group is specified.</p>
--load-balancer-name	<p>Specifies the name of a previously configured Elastic Load Balancing load balancer to associate with your service.</p> <p>Deprecated, use target-groups.</p> <p>Required: No</p>
--role	<p>Specifies the name or full Amazon Resource Name (ARN) of the IAM role that allows Amazon ECS to make calls to your load balancer or target group on your behalf. This parameter is required if you're using a load balancer or target group with your service. If you specify the role parameter, you must also specify a load balancer name or target group ARN, along with a container name and container port.</p> <p>Required: No, unless a load balancer or target group is specified.</p>
--health-check-grace-period	<p>Specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started.</p> <p>Required: No</p>

Name	Description
--create-log-groups	Creates the CloudWatch log groups specified in your Compose files. Required: No
--force-deployment	Forces a new deployment of the service. Required: No
--enable-service-discovery	Specifies whether to enable service discovery for this service. Required: No
--vpc	Specifies the VPC that will be attached to the private DNS namespace for service discovery. This parameter is required if --private-dns-namespace is specified. Required: No
--private-dns-namespace	Specifies the name of the private DNS namespace to use with service discovery. The Amazon ECS CLI automatically creates the namespace if it doesn't exist. For example, if the namespace is <code>corp</code> , a service named <code>foo</code> is reachable via DNS at <code>foo.corp</code> . If you use this parameter, you must also specify a VPC using the --vpc parameter. Required: No
--private-dns-namespace-id	Specifies the ID of an existing private DNS namespace to use with service discovery. If you use this parameter, you can't specify either --private-dns-namespace or --vpc. Required: No
--public-dns-namespace	Specifies the name of the public DNS namespace to use with service discovery. For example, if the namespace is <code>corp</code> , a service named <code>foo</code> is reachable via DNS at <code>foo.corp</code> . Required: No
--public-dns-namespace-id	Specifies the ID of an existing public DNS namespace to use with service discovery. If you use this parameter, you can't specify a --public-dns-namespace. Required: No
--sd-container-name	Specifies the name of the container, which is referred to as a service in your Docker Compose file. For more information, see Service configuration reference . This parameter is required if you're using SRV records. Required: No, unless SRV DNS records are being used.
--sd-container-port	Specifies the port on the container that will be used for service discovery. This parameter is required if you're using SRV records. Required: No, unless SRV DNS records are being used.

Name	Description
--dns-ttl	<p>Specifies the amount of time, in seconds, that you want DNS resolvers to cache the settings for the DNS records used for service discovery.</p> <p>Default value: 60</p> <p>Required: No</p>
--dns-type	<p>Specifies the type of DNS record used for service discovery. Accepted values are A or SRV. If your task uses either the bridge or host network modes, SRV records are required. If your task uses the awsvpc network mode, A records are the default.</p> <p>Required: No</p>
--healthcheck-custom-config-failure-threshold	<p>Specifies the number of 30-second intervals that you want the service discovery service to wait after receiving an <code>UpdateInstanceCustomHealthStatus</code> request before it changes the health status.</p> <p>Default value: 1</p> <p>Required: No</p>
--update-service-discovery	<p>If specified, this enables the service discovery service settings for --dns-ttl and --healthcheck-custom-config-failure-threshold to be updated.</p> <p>Required: No</p>

Name	Description
--scheduling-strategy <i>value</i>	<p>Specifies the scheduling strategy to use for the service.</p> <p>There are two service scheduler strategies available:</p> <ul style="list-style-type: none"> • 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. 533). • DAEMON—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon (p. 532). <p>Note Fargate tasks do not support the DAEMON scheduling strategy.</p> <p>For more information, see Service scheduler concepts (p. 531).</p> <p>Type: String</p> <p>Valid values: REPLICA DAEMON</p> <p>Default value: REPLICA</p> <p>Required: No</p>
--tags <i>key1=value1, key2=value2</i>	<p>Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format: key1=value1, key2=value2, key3=value3. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the --disable-ecs-managed-tags flag. For more information, see Tagging Resources (p. 130).</p> <p>Type: Key value pairs</p> <p>Required: No</p>
--disable-ecs-managed-tags	<p>Disable the Amazon ECS managed tags. For more information, see Tagging your resources for billing (p. 606).</p> <p>Required: No</p>

Name	Description
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Using a Load Balancer

You can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service. For more information, see [Service load balancing \(p. 574\)](#). After you create a service, you can't change the load balancer name or target group ARN, container name, and container port specified in the service definition.

Note

You must create your load balancer resources before you can configure a service to use them. Your load balancer resources should reside in the same VPC as your container instances, and they should be configured to use the same subnets. You must also add a security group rule to your container instance security group that allows inbound traffic from your load balancer. For more information, see [Creating a load balancer \(p. 579\)](#).

- To configure your service to use an existing Elastic Load Balancing Classic Load Balancer, you must specify the load balancer name, the container name (as it appears in a container definition), and the

container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance is registered with the load balancer specified here.

- To configure your service to use an existing Elastic Load Balancing Application Load Balancer, you must specify the load balancer target group ARN, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance and port combination is registered as a target in the target group specified here.

The `--health-check-grace-period` option specifies the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started. This is valid only if your service is configured to use a load balancer. If your tasks take a while to start and respond to Elastic Load Balancing health checks, you can specify a health check grace period of up to 1,800 seconds during which the Amazon ECS service scheduler ignores the Elastic Load Balancing health check status. This grace period can prevent the Amazon ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

Using Service Discovery

Your Amazon ECS service can optionally be configured to use Amazon ECS Service Discovery. Service discovery uses Amazon Route 53 auto naming API actions to manage DNS entries for your service's tasks, making them discoverable within your VPC. For more information, see [Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI \(p. 70\)](#).

Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

When using the `ecs-cli compose service up` command, using the `--tags` flag allows you to add metadata tags to the task definition and service. The tags will be added to the service and task definition when the resources are created. The tags will be propagated from your task definition to tasks created by the service. Amazon ECS managed tags are enabled by default if you have opted in to the new Amazon Resource Name (ARN) and resource identifier (ID) formats unless you specifically disable them using the `--disable-ecs-managed-tags` flag. For more information, see [Tagging your resources for billing \(p. 606\)](#).

Examples

Example 1

This example brings up an Amazon ECS service with the project name `hello-world` from the `hello-world.yml` compose file.

```
ecs-cli compose --project-name hello-world --file hello-world.yml service up
```

Output:

```
INFO[0000] Using ECS task definition                                     TaskDefinition="ecscompose-hello-world:7"
INFO[0000] Created an ECS service                                         service=ecscompose-service-hello-world
world taskDefinition="ecscompose-hello-world:7"
INFO[0000] Updated ECS service successfully                                 desiredCount=1
serviceName=ecscompose-service-hello-world
INFO[0015] (service ecscompose-service-hello-world) has started 1 tasks: (task
682dc22f-8bfa-4c28-b6f8-3a916bd8f86a). timestamp=2017-08-18 21:16:00 +0000 UTC
```

```

INFO[0060] Service status                                desiredCount=1 runningCount=1
  serviceName=ecscompose-service-hello-world
INFO[0060] ECS Service has reached a stable state      desiredCount=1 runningCount=1
  serviceName=ecscompose-service-hello-world

```

Example 2

This example creates a service from the `nginx-compose.yml` compose file and configures it to use an existing Application Load Balancer.

```

ecs-cli compose -f nginx-compose.yml service up --target-group-arn
  arn:aws:elasticloadbalancing:us-east-1:aws_account_id:targetgroup/ecs-cli-
alb/9856106fcc5d4be8 --container-name nginx --container-port 80 --role ecsServiceRole

```

Example 3

This example creates a service from the `nginx-compose.yml` compose file and configures it to use an existing Application Load Balancer with a health check grace period of 25 seconds.

```

ecs-cli compose -f nginx-compose.yml service up --target-group-arn
  arn:aws:elasticloadbalancing:us-east-1:aws_account_id:targetgroup/ecs-cli-
alb/9856106fcc5d4be8 --container-name nginx --container-port 80 --role ecsServiceRole --
  health-check-grace-period 25

```

ecs-cli compose service ps, list

Lists all the containers in your cluster that belong to the service created with the compose project.

Syntax

`ecs-cli compose service ps|list [--desired-status status] [--help]`

Options

Name	Description
<code>--desired-status <i>status</i></code>	The container desired status to filter the container list results with. Required: No Valid values: RUNNING STOPPED
<code>--region, -r <i>region</i></code>	Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command. Type: String Required: No
<code>--cluster-config <i>cluster_config_name</i></code>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
<code>--ecs-profile <i>ecs_profile</i></code>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.

Name	Description
	Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

ecs-cli compose service scale

Scales the desired count of the service to the specified count.

Syntax

```
ecs-cli compose service scale [--deployment-max-percent n] [--deployment-min-healthy-percent n]
[-timeout value] n [--help]
```

Options

Name	Description
--deployment-max-percent	Specifies the upper limit (as a percentage of the service's <code>desiredCount</code>) of the number of running tasks that can be running in a service during a deployment. For more information, see maximumPercent (p. 537) . Default value: 200 Required: No
--deployment-min-healthy-percent	Specifies the lower limit (as a percentage of the service's <code>desiredCount</code>) of the number of running tasks that must remain running and healthy in a service during a deployment. For more information, see minimumHealthyPercent (p. 538) . Default value: 100 Required: No
--timeout <i>value</i>	Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS

Name	Description
	<p>CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes).</p> <p>Default value: 5</p> <p>Required: No</p>
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--cluster-config <i>cluster_config_name</i>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
--ecs-profile <i>ecs_profile</i>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command.</p> <p>Type: String</p> <p>Required: No</p>
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Examples

Example 1

This example scales the service created by the `hello-world` project to a desired count of 2.

```
ecs-cli compose --project-name hello-world --file hello-world.yml service scale 2
```

Output:

```

INFO[0000] Updated ECS service successfully
serviceName=ecscompose-service-hello-world
INFO[0000] Service status
serviceName=ecscompose-service-hello-world
INFO[0030] (service ecscompose-service-hello-world) has started 1 tasks: (task
80602da8-442c-48ea-a8a9-80328c302b89). timestamp=2017-08-18 21:17:44 +0000 UTC
INFO[0075] Service status
serviceName=ecscompose-service-hello-world
INFO[0075] ECS Service has reached a stable state
serviceName=ecscompose-service-hello-world

```

ecs-cli compose service stop

Stops the running tasks that belong to the service created with the compose project. This command updates the desired count of the service to 0.

The `--timeout` option specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes).

Syntax

`ecs-cli compose service stop [--timeout value] [--help]`

Options

Name	Description
<code>--timeout <i>value</i></code>	<p>Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes).</p> <p>Default value: 5</p> <p>Required: No</p>
<code>--region, -r <i>region</i></code>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the <code>configure</code> command.</p> <p>Type: String</p> <p>Required: No</p>
<code>--cluster-config <i>cluster_config_name</i></code>	<p>Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default.</p> <p>Type: String</p> <p>Required: No</p>
<code>--ecs-profile <i>ecs_profile</i></code>	<p>Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the <code>configure profile</code> command.</p> <p>Type: String</p>

Name	Description
	Required: No
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p> <p>Required: No</p>
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>
--help, -h	Shows the help text for the specified command.
	Required: No

ecs-cli compose service rm, delete, down

Updates the desired count of the service to 0 and then deletes the service.

Syntax

This command accepts `rm`, `delete`, or `down` when used.

ecs-cli compose service rm|delete|down [--timeout *value*] [--delete-namespace] [--help]

Options

Name	Description
--timeout <i>value</i>	<p>Specifies the timeout value, in minutes (decimals supported), to wait for the running task count to change. If the running task count has not changed for the specified period of time, the Amazon ECS CLI times out and returns an error. Setting the timeout to 0 causes the command to return without checking for success. The default timeout value is 5 (minutes).</p> <p>Default value: 5</p> <p>Required: No</p>
--delete-namespace	If specified, the private namespace created with either the compose service create or compose service up commands is deleted.
--region, -r <i>region</i>	<p>Specifies the AWS Region to use. Defaults to the cluster configured using the configure command.</p> <p>Type: String</p> <p>Required: No</p>

Name	Description
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example 1

This example scales the service created by the `hello-world` project to a desired count of 0 and then deletes the service.

```
ecs-cli compose --project-name hello-world --file hello-world.yml service rm
```

Output:

```
INFO[0000] Updated ECS service successfully
  serviceName=ecscompose-service-hello-world
desiredCount=0
INFO[0000] Service status
  serviceName=ecscompose-service-hello-world
desiredCount=0 runningCount=2
INFO[0015] Service status
  serviceName=ecscompose-service-hello-world
desiredCount=0 runningCount=0
INFO[0015] (service ecscompose-service-hello-world) has stopped 2 running tasks: (task
  682dc22f-8bfa-4c28-b6f8-3a916bd8f86a) (task 80602da8-442c-48ea-a8a9-80328c302b89).
  timestamp=2017-08-18 21:25:28 +0000 UTC
INFO[0015] ECS Service has reached a stable state
  serviceName=ecscompose-service-hello-world
desiredCount=0 runningCount=0
INFO[0015] Deleted ECS service
  serviceName=ecscompose-service-hello-world
```

```
INFO[0015] ECS Service has reached a stable state           desiredCount=0 runningCount=0
serviceName=ecscompose-service-hello-world
```

ecs-cli logs

Retrieves container logs from CloudWatch Logs. Only valid for tasks that use the `awslogs` driver and have a log stream prefix specified.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli logs --task-id task_id [--task-def task_definition] [--follow]
[--filter-pattern search_string] [--since n_minutes] [--start-time
2006-01-02T15:04:05+07:00] [--end-time 2006-01-02T15:04:05+07:00] [--timestamps] [--help]
```

Options

Name	Description
<code>--task-id <i>task_id</i></code>	Prints the logs for this ECS task. Type: String Required: Yes
<code>--task-def <i>task_definition</i></code>	Specifies the name or full Amazon Resource Name (ARN) of the ECS task definition associated with the task ID. This is needed only if the task has been stopped. Type: String Required: No
<code>--follow</code>	Specifies if the logs should be streamed. Required: No
<code>--filter-pattern <i>search_string</i></code>	Specifies the substring to search for within the logs. Type: String Required: No
<code>--since <i>n</i></code>	Returns logs newer than a relative duration in minutes. Can't be used with <code>--start-time</code> . Type: Integer Required: No
<code>--start-time <i>timestamp</i></code>	Returns logs after a specific date (format: RFC 3339. Example: <code>2006-01-02T15:04:05+07:00</code>). Can't be used with <code>--since</code> flag. Required: No

Name	Description
--end-time <i>timestamp</i>	Returns logs before a specific date (format: RFC 3339. Example: 2006-01-02T15:04:05+07:00). Cannot be used with --follow. Required: No
--timestamps	Specifies if timestamps are shown on each line in the log output. Required: No
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Examples

Example

This example prints the log for a task.

```
ecs-cli logs --task-id task_id
```

The contents of the log is in the output if successful.

ecs-cli check-attributes

Checks if a given list of container instances can run a given task definition by checking their attributes. Outputs attributes that are required by the task definition but not present on the container instances.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli check-attributes [--task-def task_definition] [--container-instances value] [--help]
```

Options

Name	Description
--task-def task_definition	Specifies the name or full Amazon Resource Name (ARN) of the ECS task definition associated with the task ID. This is only needed if the task has been stopped. Type: String Required: No
--container-instances value	A list of container instance IDs or full ARN entries to check if all required attributes are available for the Task Definition to RunTask. Required: No
--region, -r region	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config cluster_config_name	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile ecs_profile	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the profile command. Type: String

Name	Description
	Required: No
--aws-profile <i>aws_profile</i>	<p>Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code>.</p> <p>Type: String</p>
	Required: No
--cluster, -c <i>cluster_name</i>	<p>Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the <code>configure</code> command.</p> <p>Type: String</p>
	Required: No
--help, -h	Shows the help text for the specified command.
	Required: No

Examples

Example

This example checks multiple container instances and verifies that they contain the attributes necessary to successfully run the specified task definition.

```
ecs-cli check-attributes --container-instances 28c5abd2-360e-41a0-81d8-0afca2d08d9b,45510138-f24f-47c6-a418-71c46dd51f88 --cluster default --region us-east-2 --task-def fluentd-test
```

Output:

Container Instance	Missing Attributes
28c5abd2-360e-41a0-81d8-0afca2d08d9b	com.amazonaws.ecs.capability.logging-driver.fluentd
45510138-f24f-47c6-a418-71c46dd51f88	None

ecs-cli registry-creds

Facilitates the creation and use of private registry credentials within Amazon ECS. For more information, see [Private registry authentication for tasks \(p. 300\)](#).

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli registry-creds [--region region] [--cluster-config cluster_config_name] [--ecs-profile ecs_profile] [--aws-profile aws_profile] [--help] [subcommand] [arguments] [--help]
```

Options

Name	Description
--region, -r <i>region</i>	Specifies the AWS Region to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--cluster-config <i>cluster_config_name</i>	Specifies the name of the Amazon ECS cluster configuration to use. Defaults to the cluster configuration set as the default. Type: String Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--cluster, -c <i>cluster_name</i>	Specifies the Amazon ECS cluster name to use. Defaults to the cluster configured using the configure command. Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Available Subcommands

The **ecs-cli registry-creds** command supports the following subcommands. Each of these subcommands has their own flags associated with them, which can be displayed using the `--help` flag.

up

Generates AWS Secrets Manager secrets and an IAM task execution role for use in an Amazon ECS task definition. For more information, see [ecs-cli registry-creds up \(p. 142\)](#).

help

Shows the help text for the specified command.

ecs-cli registry-creds up

Generates AWS Secrets Manager secrets and an IAM task execution role for use in an Amazon ECS task definition.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli registry-creds up ./creds_input_file.yml --role-name value [--update-existing-secrets]
[--no-role] [--no-output-value] [--output-dir value] [--tags key1=value1,key2=value2] [--help]
```

Options

Name	Description
<code>./creds_input_file.yml</code>	Specifies the values related to private registry authentication. For more information, see Using Private Registry Authentication (p. 143) . Required: Yes
<code>--role-name value</code>	The name to use for the new task execution role. If the role already exists, new policies are attached to the existing role. For more information, see Amazon ECS task execution IAM role (p. 691) . Note We recommend creating a new task execution role specific to each application to avoid granting permissions to your secrets for applications that do not need them. Required: Yes
<code>--update-existing-secrets</code>	Specifies whether existing secrets should be updated with new credential values. Required: No
<code>--no-role</code>	If specified, no task execution role is created. Required: No
<code>--no-output-file</code>	If specified, no output file for use with compose is created. Required: No
<code>--output-dir value</code>	The directory where the output file should be created. If none specified, the file is created in the current working directory. Required: No
<code>--tags key1=value1,key2=value2</code>	Specifies the metadata to apply to your AWS resources. Each tag consists of a key and an optional value. Tags use the following format:

Name	Description
	<p>key1=value1, key2=value2, key3=value3. For more information, see Tagging Resources (p. 144).</p> <p>Type: Key value pairs</p> <p>Required: No</p>
--help, -h	<p>Shows the help text for the specified command.</p> <p>Required: No</p>

Using Private Registry Authentication

When using the `ecs-cli registry-creds up` command to manage your private registry authentication credentials, there are certain fields that are specified using an input file. You must specify a file name or path to an input file when using this command.

Currently, the file supports the follow schema:

```
version: 1
registry_credentials:
  registry_name:
    secrets_manager_arn: string
    username: string
    password: string
    kms_key_id: string
    container_names:
      - string
```

The following are descriptions for each of these fields.

registry_name

Used as the secret name when creating a new secret or updating an existing secret. The secret name must be ASCII letters, digits, or any of the following characters: /_+=. @-. The Amazon ECS CLI adds a prefix to the secret name to indicate that it was created by the CLI. For more information, see [CreateSecret](#).

Required: No

secrets_manager_arn

The full ARN of an existing secret. Used to specify or update an existing secret. Must be in the following format:

```
arn:aws:secretsmanager:region:aws_account_id:secret:secret_name
```

Required: No

username

Specifies the user name for the private registry. We recommend using environment variables for the user name to ensure that no sensitive information is stored in the input file. When using environment variables, use the format `#{VAR_NAME}`.

Required: No

password

Specifies the password for the private registry. We recommend using environment variables for the password to ensure that no sensitive information is stored in the input file. When using environment variables, use the format `#{VAR_NAME}`.

Required: No

kms_key_id

Specifies the ARN, Key ID, or alias of the AWS KMS key (KMS key) to be used to encrypt the secret. For more information, see [CreateSecret](#).

Required: No

container_names

Corresponds to a service name in a Docker compose file. For more information, see [ecs-cli compose \(p. 102\)](#) or [ecs-cli compose service \(p. 114\)](#).

Required: No

Tagging Resources

The Amazon ECS CLI supports adding metadata in the form of resource tags to your AWS resources. Each tag consists of a key and an optional value. Resource tags can be used for cost allocation, automation, and access control. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

When using the `ecs-cli registry-creds up` command, using the `--tags` flag enables you to add metadata tags to the Secrets Manager secrets and then IAM roles.

Note

Existing Secrets Manager secrets within your account will be tagged, but IAM roles can only be tagged during creation. If you're using an existing IAM role, new tags can't be added.

Examples

Create a Secret with Private Registry Authentication Credentials

This example creates a secret with the private registry credentials specified in the `creds_input.yml` input file.

Create a private registry credentials file, named `creds_input.yml` that contains the user name and password for the private registry as well as the name of the container that will use the private registry credentials. We recommend using environment variables for the credentials to ensure that no sensitive information is stored in the input file. The container name in this file corresponds to the service name `database` in the Docker compose file.

```
version: '1'  
registry_credentials:  
  dockerhub:  
    username: ${MY_REPO_USERNAME}  
    password: ${MY_REPO_PASSWORD}  
    container_names:  
      - database
```

Important

We recommend using environment variables for the password to ensure that no sensitive information is stored in the input file. If your input file contains sensitive information, make sure that you delete it after use.

Create the secret. This command creates a secret using the name from the input file, in this example it is dockerhub. The Amazon ECS CLI adds a prefix to the secret name to indicate that it was created by the CLI. You also specify the name of your task execution role.

```
ecs-cli registry-creds up ./creds_input.yml --role-name secretsTaskExecutionRole
```

Output:

```
INFO[0000] Processing credentials for registry dockerhub...
INFO[0000] New credential secret created:
arn:aws:secretsmanager:region:aws_account_id:secret:amazon-ecs-cli-setup-dockerhub-VeDqXm
INFO[0000] Creating resources for task execution role ecsTaskExecutionRole...
INFO[0000] Created new task execution role arn:aws:iam::aws_account_id:role/
ecsTaskExecutionRole
INFO[0000] Created new task execution role policy arn:aws:iam::aws_account_id:policy/
amazon-ecs-cli-setup-bugBashRole-policy-20181023T210805Z
INFO[0000] Attached AWS managed policy arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy to role ecsTaskExecutionRole
INFO[0001] Attached new policy arn:aws:iam::aws_account_id:policy/amazon-ecs-cli-setup-
bugBashRole-policy-20181023T210805Z to role ecsTaskExecutionRole
INFO[0001] Writing registry credential output to new file C:\Users\brandejo\regcreds
\regCredTest\ecs-registry-creds_20181023T210805Z.yml
```

An output file is created by this command that contains the task execution role name, the ARN of the secret that was created, and the container name. This file is specified using the `--registry-creds` option when using either the `ecs-cli compose` or `ecs-cli compose service` commands. For more information, see [ecs-cli compose \(p. 102\)](#) or [ecs-cli compose service \(p. 114\)](#).

The following is an example output file:

```
version: "1"
registry_credential_outputs:
  task_execution_role: secretsTaskExecutionRole
  container_credentials:
    dockerhub:
      credentials_parameter: arn:aws:secretsmanager:region:aws_account_id:secret:amazon-
      ecs-cli-setup-dockerhub-bbHiEk
      container_names:
        - database
```

Create a Secret with Private Registry Authentication Credentials That Use a KMS key

This example creates a secret with the private registry credentials that are encrypted using a KMS key specified in the `creds_input.yml` input file.

Create a private registry credentials file, named `creds_input.yml` that contains the user name and password for the private registry as well as the name of the container that will use the private registry credentials. We recommend using environment variables for the credentials to ensure that no sensitive information is stored in the input file. The specified KMS key ARN encrypts the values when storing the secret. The container name in this file corresponds to the service name `database` in the Docker compose file.

```
version: '1'
registry_credentials:
  dockerhub:
    username: ${MY_REPO_USERNAME}
    password: ${MY_REPO_PASSWORD}
    kms_key_id: kmsKeyARN
```

```
container_names:  
  - database
```

Important

We recommend using environment variables for the password to ensure that no sensitive information is stored in the input file. If your input file contains sensitive information, make sure that you delete it after use.

Create Multiple Secrets For Multiple Private Registries

This example creates multiple secrets with the private registry credentials for multiple registries.

Create a private registry credentials file, named `creds_input.yml` that contains the credentials from two different private registries. Each set of credentials are used to create its own secret. This example also shows two different containers using one secret.

```
version: '1'  
registry_credentials:  
  dockerhub:  
    username: ${MY_REPO_USERNAME}  
    password: ${MY_REPO_PASSWORD}  
    container_names:  
      - prod  
      - dev  
  quay.io:  
    username: ${MY_REPO_USERNAME}  
    password: ${MY_REPO_PASSWORD}  
    container_names:  
      - database
```

Important

We recommend using environment variables for the password to ensure that no sensitive information is stored in the input file. If your input file contains sensitive information, make sure that you delete it after use.

ecs-cli local

Runs your Amazon ECS tasks locally by creating a Docker Compose file from an Amazon ECS task definition.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli local [subcommand] [arguments] [--help]
```

Options

Name	Description
--region, -r region	Specifies the AWS Region to use. Defaults to the Region configured using the <code>configure</code> command. Type: String

Name	Description
	Required: No
--ecs-profile <i>ecs_profile</i>	Specifies the name of the Amazon ECS profile configuration to use. Defaults to the profile configured using the configure profile command. Type: String Required: No
--aws-profile <i>aws_profile</i>	Specifies the AWS profile to use. Enables you to use the AWS credentials from an existing named profile in <code>~/.aws/credentials</code> . Type: String Required: No
--help, -h	Shows the help text for the specified command. Required: No

Available Subcommands

The **ecs-cli local** command supports the following subcommands. Each of these subcommands has their own flags associated with them, which can be displayed using the `--help` flag.

create

Creates a Docker Compose file from an Amazon ECS task definition. For more information, see [ecs-cli local create \(p. 147\)](#).

up

Runs containers locally from an Amazon ECS task definition. For more information, see [ecs-cli local up \(p. 149\)](#).

down

Stops and removes locally running containers. For more information, see [ecs-cli local down \(p. 151\)](#).

ps

Lists locally running containers. For more information, see [ecs-cli local ps \(p. 152\)](#).

help

Shows the help text for the specified command.

ecs-cli local create

Creates a Docker Compose file from an Amazon ECS task definition.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli local create [--task-def-file filename] [--task-def-remote value] [--force] [--output output_file]
```

Options

Name	Description
--task-def-file <i>filename</i>	Specifies the filename that contains the task definition JSON to convert to a Docker Compose file. If one is not specified, the ECS CLI will look for a file named <code>task-definition.json</code> in the current directory. Type: JSON Required: No
--task-def-remote <i>value</i>	Specifies the full Amazon Resource Name (ARN) or family:revision of the task definition to convert to a Docker Compose file. If you specify a task definition family without a revision, the latest revision is used. Type: string Required: No
--force	Overwrites any existing Docker Compose output file without prompting for confirmation.
--output <i>output_file</i>	Specifies the local filename to write the Docker Compose file to. If one is not specified, the default is <code>docker-compose.local.yml</code> . Type: string Required: No
--help, -h	Shows the help text for the specified command. Required: No

Examples

Create a Docker Compose file from a local JSON file

This example creates a Docker Compose file from a local JSON file containing an Amazon ECS task definition.

```
ecs-cli local create --task-def-file task-definition.json
```

Output:

```
INFO[0000] Successfully wrote docker-compose.ecs-local.yml
INFO[0000] Successfully wrote docker-compose.ecs-local.override.yml
```

Create a Docker Compose file from a remote task definition

This example creates a Docker Compose file from the latest revision of an Amazon ECS task definition named `hello-world`.

```
ecs-cli local create --task-def-remote hello-world
```

Output:

```
INFO[0000] Successfully wrote docker-compose.ecs-local.yml
INFO[0000] Successfully wrote docker-compose.ecs-local.override.yml
```

ecs-cli local up

Runs containers locally from an Amazon ECS task definition. By default this command looks for a task definition JSON file named `task-definition.json` in the current directory. If the task definition file does not exist, then one must be specified using one of the `--task-def` options described below. This command also creates a local Docker Compose file as specified in the `--output` option prior to running the containers.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli local up [--task-def-compose filename] [--task-def-file filename] [--task-def-remote value] [--force] [--output output_file] [--override filename]
```

Options

Name	Description
<code>--task-def-compose <i>filename</i></code>	Specifies the Docker Compose file to run locally. Type: string Required: No
<code>--task-def-file <i>filename</i></code>	Specifies the task definition JSON file to run locally. If one is not specified, the ECS CLI will look for a file named <code>task-definition.json</code> in the current directory. Type: string Required: No
<code>--task-def-remote <i>value</i></code>	Specifies the full Amazon Resource Name (ARN) or family:revision of the task definition to convert to a Docker Compose file. If you specify a task definition family without a revision, the latest revision is used. Type: string Required: No
<code>--force</code>	Overwrites any existing Docker Compose output file without prompting for confirmation.

Name	Description
--output <i>output_file</i>	<p>Specifies the local filename to write the Docker Compose file to. If one is not specified, the default is <code>docker-compose.local.yml</code>. If the output file already exists, the CLI will prompt you with an overwrite request.</p> <p>Type: string</p> <p>Required: No</p>
--override <i>filename</i>	<p>Specifies the local Docker Compose override filename to use.</p> <p>Type: string</p> <p>Required: No</p>

Examples

Run containers locally from a local task definition JSON file

This example runs the containers locally that are defined in a local task definition file named `hello-world.json`.

```
ecs-cli local create --task-def-file hello-world.json
```

Output:

```
INFO[0001] Successfully wrote docker-compose.ecs-local.yml
INFO[0002] Successfully wrote docker-compose.ecs-local.override.yml
INFO[0002] The network ecs-local-network already exists
INFO[0002] The amazon-ecs-local-container-endpoints container already exists with ID
5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1
INFO[0002] Started container with ID
5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1
INFO[0002] Using docker-compose.ecs-local.yml, docker-compose.ecs-local.override.yml files
to start containers
Compose out: Found orphan containers (downloads_httpd_1) for this project. If you removed
or renamed this service in your compose file, you can run this command with the --remove-
orphans flag to clean it up.
Creating downloads_simple-app_1 ... done
```

Run containers locally from a remote task definition

This example runs the containers locally that are defined in the latest revision of an Amazon ECS task definition named `hello-world`.

```
ecs-cli local up --task-def-remote hello-world
```

Output:

```
INFO[0000] Reading task definition from hello-world

INFO[0002] Successfully wrote docker-compose.ecs-local.yml
INFO[0004] Successfully wrote docker-compose.ecs-local.override.yml
INFO[0004] The network ecs-local-network already exists
INFO[0005] The amazon-ecs-local-container-endpoints container already exists with ID
5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1
```

```
INFO[0005] Started container with ID  
5976522f4cafb840e5f003a2285fc439ed1b2a89aa74634958c6a6105ca6edd1  
INFO[0005] Using docker-compose.ecs-local.yml, docker-compose.ecs-local.override.yml files  
to start containers  
Compose out: Found orphan containers (downloads_httpd_1) for this project. If you removed  
or renamed this service in your compose file, you can run this command with the --remove-  
orphans flag to clean it up.  
Creating downloads_hello-world_1 ... done
```

ecs-cli local down

Stops and removes locally running containers.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli local down [--task-def-file filename] [--task-def-remote task_definition_ARN_family]  
[--all]
```

Options

Name	Description
--task-def-file <i>filename</i>	Stop and remove all running containers matching the task definition filename. If both --task-def-file and --task-def-remote are omitted, the ECS CLI defaults to task-definition.json. Type: string Required: No
--task-def-remote <i>value</i>	Stops and remove all running containers matching the specified task definition Amazon Resource Name (ARN) or family:revision. If you specify a task definition family without a revision, the latest revision is used. Type: string Required: No
--all	Stops and removes all locally running containers. Type: string Required: No

Examples

Stop a locally running container

This example stops a locally running container that is using the hello-world.json task definition file.

```
ecs-cli local down --task-def-file hello-world.json
```

Output:

```
INFO[0000] Stop and remove 1 container(s)
INFO[0011] Stopped container with id 9df4c584d905
INFO[0011] Removed container with id 9df4c584d905
INFO[0011] The network ecs-local-network has no more running tasks
INFO[0012] Stopped container with name amazon-ecs-local-container-endpoints
INFO[0012] Removed container with name amazon-ecs-local-container-endpoints
INFO[0012] Removed network with name ecs-local-network
```

Stop all locally running containers

This example stops all locally running containers.

```
ecs-cli local down --all
```

ecs-cli local ps

Lists locally running containers.

Important

Some features described might only be available with the latest version of the Amazon ECS CLI. For more information about obtaining the latest version, see [Installing the Amazon ECS CLI \(p. 55\)](#).

Syntax

```
ecs-cli local ps [--task-def-file filename] [--task-def-remote value] [--all] [--json]
```

Options

Name	Description
--task-def-file <i>filename</i>	<p>Lists all locally running containers matching the task definition filename. If both --task-def-file and --task-def-remote are omitted, the ECS CLI defaults to <code>task-definition.json</code>.</p> <p>Type: string</p> <p>Required: No</p>
--task-def-remote <i>value</i>	<p>Lists all running containers matching the task definition Amazon Resource Name (ARN) or family:revision. If you specify a task definition family without a revision, the latest revision is used.</p> <p>Type: string</p> <p>Required: No</p>
--all	<p>Lists all locally running containers.</p> <p>Type: string</p> <p>Required: No</p>
--json	Sets the output to JSON format.

Name	Description
	Type: string Required: No

Examples

List all locally running containers

This example lists all locally running containers.

```
ecs-cli local ps --all
```

Output:

CONTAINER ID	IMAGE	STATUS	PORTS	NAMES
TASKDEFINITION				
9df4c584d905	httpd:2.4	Up 15 seconds	0.0.0.0:80->80/tcp	/
downloads_simple-app_1	/Users/brandejo/Downloads/task-definition.json			

List all locally running containers using a specified task definition file

This example lists the locally running containers using the `hello-world.json` task definition.

```
ecs-cli local ps --task-def-file hello-world.json
```

Using Docker Compose File Syntax

The `ecs-cli compose` and `ecs-cli compose service` commands allow you to create task definitions and manage your Amazon ECS tasks and services using Docker Compose files. For more information, see [ecs-cli compose \(p. 102\)](#) and [ecs-cli compose service \(p. 114\)](#).

At this time, the latest version of the Amazon ECS CLI only supports the major versions of [Docker Compose file syntax](#) versions 1, 2, and 3. The version specified in the compose file must be the string "1", "1.0", "2", "2.0", "3", or "3.0". Docker Compose minor versions are not supported.

By default, the Amazon ECS CLI commands look for a Docker Compose file in the current directory, named `docker-compose.yml`. However, you can also specify a different file name or path to a Compose file with the `--file` option. This is especially useful for managing tasks and services from multiple Compose files at a time with the Amazon ECS CLI.

The following parameters are supported in Compose files for the Amazon ECS CLI:

- `cap_add` (not valid for tasks using the Fargate launch type)
- `cap_drop` (not valid for tasks using the Fargate launch type)
- `command`
- `cpu_shares`

Note

If you're using the Compose version 3.0 format, `cpu_shares` should be specified in the `ecs-params.yml` file. For more information, see [Using Amazon ECS Parameters \(p. 155\)](#).

- `devices` (not valid for tasks using the Fargate launch type)
- `dns`
- `dns_search`

- `entrypoint`
- `environment`: If an environment variable value isn't specified in the Compose file, but it exists in the shell environment, the shell environment variable value is passed to the task definition that is created for any associated tasks or services.

Important

We don't recommend using plaintext environment variables for sensitive information, such as credential data.

- `env_file`

Important

We don't recommend using plaintext environment variables for sensitive information, such as credential data.

- `extends` (Compose file version 1.0 and 2 only)
- `extra_hosts`
- `healthcheck` (Compose file version 3.0 only)

Note

The `start_period` field isn't supported using the Compose file. To specify a `start_period`, use the `ecs-params.yml` file. For more information, see [Using Amazon ECS Parameters \(p. 155\)](#).

- `hostname`
- `image`
- `labels`
- `links` (not valid for tasks using the Fargate launch type)
- `log_driver` (Compose file version 1.0 only)
- `log_opt` (Compose file version 1.0 only)
- `logging` (Compose file version 2.0 and 3.0)
 - `driver`
 - `options`
- `mem_limit` (in bytes)

Note

If you're using the Compose version 3.0 format, `mem_limit` should be specified in the `ecs-params.yml` file. For more information, see [Using Amazon ECS Parameters \(p. 155\)](#).

- `mem_reservation` (in bytes)

Note

If you're using the Compose version 3.0 format, `mem_reservation` should be specified in the `ecs-params.yml` file. For more information, see [Using Amazon ECS Parameters \(p. 155\)](#).

- `ports`
- `privileged` (not valid for tasks using the Fargate launch type)
- `read_only`
- `security_opt`
- `shm_size` (Compose file version 1.0 and 2 only and not valid for tasks using the Fargate launch type)
- `tmpfs` (not valid for tasks using the Fargate launch type)
- `tty`
- `ulimits`
- `user`
- `volumes`
- `volumes_from` (Compose file version 1.0 and 2 only)
- `working_dir`

Important

The build directive isn't supported at this time.

For more information about Docker Compose file syntax, see the [Compose file reference](#) in the Docker documentation.

Using Amazon ECS Parameters

When using the `ecs-cli compose` or `ecs-cli compose service` commands to manage your Amazon ECS tasks and services, there are certain fields in an Amazon ECS task definition that do not correspond to fields in a Docker compose file. You can specify those values using an ECS parameters file with the `--ecs-params` flag. By default, the command looks for an ECS parameters file in the current directory named `ecs-params.yml`. However, you can also specify a different file name or path to an ECS parameters file with the `--ecs-params` option.

Currently, the file supports the follow schema:

```
version: 1
task_definition:
  ecs_network_mode: string
  task_role_arn: string
  task_execution_role: string
  task_size:
    cpu_limit: string
    mem_limit: string
  pid_mode: string
  ipc_mode: string
  services:
    <service_name>:
      essential: boolean
      depends_on:
        - container_name: string
          condition: string
      repository_credentials:
        credentials_parameter: string
      cpu_shares: integer
      mem_limit: string
      mem_reservation: string
      gpu: string
      init_process_enabled: boolean
      healthcheck:
        test: [ "CMD", "curl -f http://localhost" ]
        interval: string
        timeout: string
        retries: integer
        start_period: string
      firelens_configuration:
        type: string
        options:
          enable-ecs-log-metadata: boolean
      secrets:
        - value_from: string
          name: string
    docker_volumes:
      - name: string
        scope: string
        autoprovion: boolean
        driver: string
        driver_opts: string
        string: string
      labels:
        string: string
    efs_volumes:
```

```

- name: string
  filesystem_id: string
  root_directory: string
  transit_encryption: string
  transit_encryption_port: int64
  access_point: string
  iam: string
placement_constraints:
- type: string
  expression: string
run_params:
network_configuration:
awsvpc_configuration:
  subnets:
    - subnet_id1
    - subnet_id2
  security_groups:
    - secgroup_id1
    - secgroup_id2
  assign_public_ip: ENABLED
task_placement:
  strategy:
    - type: string
      field: string
constraints:
  - type: string
    expression: string
service_discovery:
  container_name: string
  container_port: integer
  private_dns_namespace:
    vpc: string
    id: string
    name: string
    description: string
  public_dns_namespace:
    id: string
    name: string
  service_discovery_service:
    name: string
    description: string
    dns_config:
      type: string
      ttl: integer
  healthcheck_custom_config:
    failure_threshold: integer

```

The fields listed under `task_definition` correspond to fields to be included in your Amazon ECS task definition.

- `ecs_network_mode` – Corresponds to `networkMode` in an ECS task definition. Supported values are `none`, `bridge`, `host`, or `awsvpc`. The default value is `bridge`. If you are using task networking, this field must be set to `awsvpc`. For more information, see [Network mode \(p. 211\)](#).
- `task_role_arn` – The name or full ARN of an IAM role to be associated with the task. For more information, see [Task role \(p. 210\)](#).
- `task_execution_role` – The name or full ARN of the task execution role. This is a required field if you want your tasks to be able to store container application logs in CloudWatch or allow your tasks to pull container images from Amazon ECR. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).
- `task_size` – The CPU and memory values for the task. If you are using the EC2 launch type, this field is optional and any value can be used. If using the Fargate launch type, this field is required and you must use one of the following sets of values for the `cpu` and `memory` parameters.

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

For more information, see [Task size \(p. 212\)](#).

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

If the `host` PID mode is used, be aware that there is a heightened risk of undesired process namespace expose. For more information, see [Docker security](#).

Note

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

- `ipc_mode` – The IPC resource namespace to use for the containers in the task. The valid values are `host`, `task`, or `none`. If `host` is specified, then all containers within the tasks that specified the `host` IPC mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If `task` is specified, all containers within the specified task share the same IPC resources. If `none` is specified, then IPC resources within the containers of a task are private and not shared with other containers in a task or on the container instance. If no value is specified, then the IPC resource namespace sharing depends on the Docker daemon setting on the container instance. For more information, see [IPC settings](#) in the *Docker run reference*.

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

If you are setting namespaced kernel parameters using `systemControls` for the containers in the task, the following will apply to your IPC resource namespace. For more information, see [System Controls](#) in the *Amazon Elastic Container Service Developer Guide*.

- For tasks that use the `host` IPC mode, IPC namespace related `systemControls` are not supported.
- For tasks that use the `task` IPC mode, IPC namespace related `systemControls` will apply to all containers within a task.

Note

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

- `services` – Corresponds to the services listed in your Docker compose file, with `service_name` matching the name of the container to run. Its fields are merged into a container definition.
- `essential` – If the `essential` parameter of a container is marked as `true`, and that container fails or stops for any reason, all other containers that are part of the task are stopped. If the

essential parameter of a container is marked as `false`, then its failure does not affect the rest of the containers in a task. The default value is `true`.

All tasks must have at least one essential container. If you have an application that is composed of multiple containers, you should group containers that are used for a common purpose into components, and separate the different components into multiple task definitions.

- `depends_on` – This parameter maps to the `dependsOn` task definition parameter. It is used to specify a list of container dependencies, which can be used for conditional startup of dependent containers or ensuring the order of startup when using multiple containers. For a task definition example, see [Example: Container dependency \(p. 321\)](#).
- `repository_credentials` – If you are using a private repository for pulling images, `repository_credentials` allows you to specify an AWS Secrets Manager secret ARN for the name of the secret containing your private repository credentials as a `credential_parameter`. For more information, see [Private registry authentication for tasks \(p. 300\)](#).
- `cpu_shares` – This parameter maps to `cpu_shares` in the [Docker compose file reference](#). If you are using Docker compose version 3, this field is optional and must be specified in the ECS params file rather than the compose file. In Docker compose version 2, this field can be specified in either the compose or ECS params file. If it is specified in the ECS params file, the value overrides the value present in the compose file.
- `mem_limit` – This parameter maps to `mem_limit` in the [Docker compose file reference](#). If you are using Docker compose version 3, this field is optional and must be specified in the ECS params file rather than the compose file. In Docker compose version 2, this field can be specified in either the compose or ECS params file. If it is specified in the ECS params file, the value overrides the value present in the compose file.
- `mem_reservation` – This parameter maps to `mem_reservation` in the [Docker compose file reference](#). If you are using Docker compose version 3, this field is optional and must be specified in the ECS params file rather than the compose file. In Docker compose version 2, this field can be specified in either the compose or ECS params file. If it is specified in the ECS params file, the value overrides the value present in the compose file.
- `gpu` – The number of physical GPUs the Amazon ECS container agent will reserve for the container. This parameter maps to the `resourceRequirements` field in a task definition. For more information, see [Working with GPUs on Amazon ECS \(p. 250\)](#).
- `init_process_enabled` – This parameter enables you to run an `init` process inside the container that forwards signals and reaps processes. This parameter maps to the `--init` option to [docker run](#).

This parameter requires version 1.25 of the Docker Remote API or greater on your container instance.

- `healthcheck` – This parameter maps to `healthcheck` in the [Docker compose file reference](#). The `test` field can also be specified as `command` and must be either a string or a list. If it's a list, the first item must be either `NONE`, `CMD`, or `CMD-SHELL`. If it's a string, it's equivalent to specifying `CMD-SHELL` followed by that string. The `interval`, `timeout`, and `start_period` fields are specified as durations in a string format. For example: `2.5s`, `10s`, `1m30s`, `2h23m`, or `5h34m56s`.

Note

If no units are specified, seconds are assumed. For example, you can specify either `10s` or simply `10`.

- `firelens_configuration` – This parameter allows you to define a log configuration using the `awsfirelens` log driver. This is used to route logs to an AWS service or partner destination for log storage and analytics. For more information, see [Custom log routing \(p. 289\)](#).
 - `type` – The log router type to use. Supported options are `fluentbit` and `fluentd`.
 - `options` – The log router options to use. This will depend on the destination you are routing your logs to. For more information, see [Custom log routing \(p. 289\)](#).
 - `secrets` – This parameter allows you to inject sensitive data into your containers by storing your sensitive data in AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. For more information, see [Specifying sensitive data \(p. 303\)](#).

- **value_from** – This is the AWS Systems Manager Parameter Store ARN or name to expose to the container. If the Systems Manager Parameter Store parameter exists in the same Region as the task you are launching, then you can use either the full ARN or name of the secret. If the parameter exists in a different Region, then the full ARN must be specified.
- **name** – The value to set as the environment variable on the container.
- **docker_volumes** – This parameter allows you to create docker volumes. The **name** key is required, and **scope**, **autoprovision**, **driver**, **driver_opts** and **labels** correspond with the Docker volume configuration fields in a task definition. For more information, see [DockerVolumeConfiguration](#) in the *Amazon Elastic Container Service API Reference*. Volumes defined with the **docker_volumes** key can be referenced in your compose file by name, even if they were not also specified in the compose file.
- **efs_volumes** – This parameter enables you to mount Amazon EFS file systems. The **name** and **filesystem_id** keys are required.
 - **name** – The name of the volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. This name is referenced in the **sourceVolume** parameter of container definition **mountPoints**.
 - **filesystem_id** – The ID of the file system for which to create the mount target.
 - **root_directory** – The directory within the Amazon EFS file system to mount as the root directory inside the host. If this parameter is omitted, the root of the Amazon EFS volume will be used. Specifying `/` will have the same effect as omitting this parameter.
 - **transit_encryption** – Whether or not to enable encryption for Amazon EFS data in transit between the Amazon ECS host and the Amazon EFS server. This parameter is required if IAM is enabled or an access point ID is specified. Valid values are `ENABLED` or `DISABLED`. `DISABLED` is the default.
 - **transit_encryption_port** – The port to use when sending encrypted data between the Amazon ECS host and the Amazon EFS server. If you do not specify a transit encryption port, it will use the port selection strategy that the Amazon EFS mount helper uses. This parameter is required if **transit_encryption** is enabled.
 - **access_point** – The ID of the Amazon EFS access point to use. If an access point is specified, the **root_directory** value will be relative to the directory set for the access point. If specified, **transit_encryption** must be enabled.
 - **iam** – Whether or not to use the Amazon ECS task IAM role defined in a task definition when mounting the Amazon EFS file system. If enabled, **transit_encryption** must be enabled in the **EFSVolumeConfiguration**. Valid values are `ENABLED` or `DISABLED`. `DISABLED` is the default.
- **placement_constraints** – This parameter allows you to specify a list of constraints on task placement within the task definition. For more information, see [TaskDefinitionPlacementConstraint](#) in the *Amazon Elastic Container Service API Reference*. It is optional if you are using the EC2 launch type. It is not supported if using the Fargate launch type.

The fields listed under `run_params` are for values needed as options to any API calls not specifically related to a task definition, such as `compose up` (`RunTask`) and `compose service up` (`CreateService`).

- **network_configuration** – Required if you specified `awsvpc` for `ecs_network_mode`. It uses one nested parameter, `awsvpc_configuration`, which has the following subfields:
 - **subnets** – A list of subnet IDs used to associate with your tasks. The listed subnets must be in the same VPC and Availability Zone as the instances on which to launch your tasks.
 - **security_groups** – A list of security group IDs to associate with your tasks. The listed security groups must be in the same VPC as the instances on which to launch your tasks.
 - **assign_public_ip** – The supported values for this field are `ENABLED` or `DISABLED`. This field is only used for tasks using the Fargate launch type. If this field is present in tasks using task networking with the EC2 launch type, the request fails.

- **task_placement** – This parameter allows you to specify task placement options. It is optional if you are using the EC2 launch type. It is not supported if using the Fargate launch type. For more information, see [Amazon ECS task placement \(p. 513\)](#).

It has the following subfields:

- **strategy** – A list of objects, with two keys. Valid keys are `type` and `field`.
 - `type` – Valid values are `random`, `binpack`, or `spread`. If `random` is specified, the `field` key should not be provided.
 - `field` – Valid values depend on the `strategy` type.
 - For `spread`, valid values are `instanceId`, `host`, or attribute key-value pairs, for example `attribute:ecs.instance-type =~ t2.*`.
 - For `binpack`, valid values are `cpu` or `memory`.
- **constraints** – A list of objects, with two keys. Valid keys are `type` and `expression`.
 - `type` – Valid values are `distinctInstance` and `memberOf`. If `distinctInstance` is specified, the `expression` key should not be provided.
 - `expression` – When `type` is `memberOf`, valid values are key-value pairs for attributes or task groups, for example `task:group == databases` or `attribute:color =~ green`.
- **service_discovery** – This parameter allows you to configure Amazon ECS Service Discovery using Amazon Route 53 auto naming API actions to manage DNS entries for your service's tasks. For more information, see [Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI \(p. 70\)](#).

Amazon ECS on AWS Fargate

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

When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

Fargate offers platform versions for Amazon Linux 2 and Microsoft Windows 2019 Server Full and Core editions. Unless otherwise specified, the information on this page applies to all Fargate platforms.

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

For information about the Regions that support Linux containers on Fargate, see [the section called "Supported Regions for Linux containers on AWS Fargate" \(p. 614\)](#).

For information about the Regions that support Windows containers on Fargate, see [the section called "Supported Regions for Windows containers on AWS Fargate" \(p. 615\)](#).

Task definitions

Amazon ECS tasks on Fargate do not support all of the task definition parameters that are available. Some parameters are not supported at all, and others behave differently for Fargate tasks.

The following task definition parameters are not valid in Fargate tasks:

- `disableNetworking`
- `dnsSearchDomains`
- `dnsServers`
- `dockerSecurityOptions`
- `extraHosts`
- `gpu`
- `ipcMode`
- `links`
- `pidMode`
- `placementConstraints`
- `privileged`
- `systemControls`

The following task definition parameters are valid in Fargate tasks, but have limitations that should be noted:

- `linuxParameters` – When specifying Linux-specific options that are applied to the container, for capabilities the `add` parameter is not supported. The `devices`, `sharedMemorySize`, and `tmpfs` parameters are not supported. For more information, see [Linux parameters \(p. 232\)](#).

- **volumes** – Fargate tasks only support bind mount host volumes, so the `dockerVolumeConfiguration` parameter is not supported. For more information, see [Volumes \(p. 241\)](#).
- **cpu** - For Windows containers on Fargate, the value cannot be less than 1 vCPU.

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

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

Network mode

Amazon ECS task definitions for Fargate require that the network mode is set to `awsvpc`. The `awsvpc` network mode provides each task with its own elastic network interface. For more information, see [Fargate task networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

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 Operating Systems

When you configure a task and container definition for Fargate, you must specify the Operating System that the container runs. The following Operating Systems are supported for Fargate:

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

Task CPU and memory

Amazon ECS task definitions for Fargate require that you specify CPU and memory at the task level. Although you can also specify CPU and memory at the container level for Fargate tasks, this is optional. Most use cases are satisfied by only specifying these resources at the task level. The table below shows the valid combinations of task-level CPU and memory.

CPU value	Memory value	Operating systems supported for Fargate
256 (.25 vCPU)	512 MB, 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

Task resource limits

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

Amazon ECS task definitions for Windows on Fargate do not support the `ulimits` parameter to define the resource limits to set for a container.

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

The following is an example task definition snippet that shows how to define a custom `nofile` limit that has been doubled:

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

For more information on the other resource limits that can be adjusted, see [Resource limits \(p. 231\)](#).

Logging

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

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

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

For more information about using the `awslogs` log driver in a task definition to send your container logs to CloudWatch Logs, see [Using the awslogs log driver \(p. 283\)](#).

For more information about the `firelens` log driver in a task definition, see [Custom log routing \(p. 289\)](#).

For more information about using the `splunk` log driver in a task definition, see [Example: splunk log driver \(p. 319\)](#).

Amazon ECS task execution IAM role

There is an optional task execution IAM role that you can specify with Fargate to allow your Fargate tasks to make API calls to Amazon ECR. The API calls pull container images as well as calling CloudWatch to store container application logs. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

Example Amazon Linux 2 task definition

The following is an example task definition that sets up a web server using the Fargate launch type with an Amazon Linux 2 operating system:

```
{
    "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",
        "platformFamily": "LINUX",
        "memory": "512",
        "networkMode": "awsvpc",
        "requiresCompatibilities": [
            "FARGATE"
        ]
    }
}
```

Example Windows task definition

The following is an example task definition that sets up a web server using the Fargate launch type with a Windows 2019 Server operating system.

```
{
    "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> </div></body></html>' > index.html && iisreset
            ],
            "entryPoint": [
                "cmd",
                "/c"
            ],
            "essential": true,
            "image": "microsoft/windowsservercore:1903",
            "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",
        "platformFamily": "WINDOWS",
        "memory": "512",
        "networkMode": "awsvpc",
        "requiresCompatibilities": [
            "FARGATE"
        ]
    }
}
```

```
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"
]
}
```

Task storage

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

- Amazon EFS volumes for persistent storage. For more information, see [Amazon EFS volumes \(p. 257\)](#).
- Bind mounts for ephemeral storage. For more information, see [Bind mounts \(p. 268\)](#).

Tasks and services

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

Task networking

Amazon ECS tasks for Fargate require the awsvpc network mode, which provides each task with an elastic network interface. When you run a task or create a service with this network mode, you must

specify one or more subnets to attach the network interface and one or more security groups to apply to the network interface.

If you are using public subnets, decide whether to provide a public IP address for the network interface. For a Fargate task in a public subnet to pull container images, a public IP address needs to be assigned to the task's elastic network interface, with a route to the internet or a NAT gateway that can route requests to the internet. For a Fargate task in a private subnet to pull container images, you need a NAT gateway in the subnet to route requests to the internet. When you host your container images in Amazon ECR, you can configure Amazon ECR to use an interface VPC endpoint. In this case, the task's private IPv4 address is used for the image pull. For more information about Amazon ECR interface endpoints, see [Amazon ECR interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon Elastic Container Registry User Guide*.

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

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

Service load balancing

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

Amazon ECS services on 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 \(p. 576\)](#).

When you create a target group for these services, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance. For more information, see [Service load balancing \(p. 574\)](#).

Using a Network Load Balancer to route UDP traffic to your Amazon ECS on Fargate tasks is only supported when using platform version 1.4 or and for tasks launched in the following Regions:

- US East (N. Virginia) - `us-east-1`
- US West (Oregon) - `us-west-2`
- EU (Ireland) - `eu-west-1`
- Asia Pacific (Tokyo) - `ap-northeast-1`

Private registry authentication

Amazon ECS tasks for Fargate can authenticate with private image registries, including Docker Hub, using basic authentication. When you enable private registry authentication, you can use private Docker images in your task definitions.

To use private registry authentication, you create a secret with AWS Secrets Manager containing the credentials for your private registry. Then, within your container definition, you specify `repositoryCredentials` with the full ARN of the secret that you created. The following snippet of a task definition shows the required parameters:

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

For more information, see [Private registry authentication for tasks \(p. 300\)](#).

Clusters

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

For more information about Amazon ECS clusters, including a walkthrough for creating a cluster, see [Amazon ECS clusters \(p. 175\)](#).

Fargate Spot

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

Windows containers on Fargate cannot use the Fargate Spot capacity provider.

With Fargate Spot you can run interruption tolerant Amazon ECS tasks at a discounted rate compared to the Fargate price. Fargate Spot runs tasks on spare compute capacity. When AWS needs the capacity back, your tasks will be interrupted with a two-minute warning. For more information, see [AWS Fargate capacity providers \(p. 180\)](#).

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 \(p. 613\)](#).

For more information about AWS Fargate usage metrics, see [Fargate usage metrics in the Amazon Elastic Container Service User Guide for 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 more information, see [Task maintenance in the Amazon Elastic Container Service User Guide for AWS Fargate](#).

The following table describes these scenarios.

Task type	Issue	Action
Standalone task	Host issue	A task retirement notice is sent using your AWS Personal Health Dashboard 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 AWS Personal Health Dashboard 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 will launch a new task in an attempt to maintain the service's desired count. No notification is sent.
	Security vulnerability	A task retirement notice is sent using your AWS Personal Health Dashboard and email. If no action is taken by the task retirement date, AWS stops the task and the service scheduler will launch a new task in an attempt to maintain the service's desired count.

Savings plans

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

To create a Savings Plan for your Fargate usage, use the **Compute Savings Plans** type. To get started, see [Getting started with Savings Plans](#) in the [Savings Plans User Guide](#).

Windows containers on 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” \(p. 283\)](#).

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
- ECS Exec
- 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 \(p. 268\)](#).

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 \(p. 169\)](#)
- [Windows platform versions \(p. 173\)](#)

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 \(p. 559\)](#).
- 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 \(p. 173\)](#).

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 \(p. 303\)](#).
 - Specify environment variables in bulk using the `environmentFiles` container definition parameter. For more information, see [Specifying environment variables \(p. 315\)](#).
 - 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 \(p. 574\)](#).
- Beginning on May 28, 2020, any new Amazon ECS task launched on Fargate using platform version `1.4.0` will have its ephemeral storage encrypted with an AES-256 encryption algorithm using an AWS owned encryption key. For more information, see [Fargate task storage \(p. 256\)](#).
- Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see [Amazon EFS volumes \(p. 257\)](#).
- The ephemeral task storage has been increased to a minimum of 20 GB for each task. For more information, see [Fargate task storage \(p. 256\)](#).
- The network traffic behavior to and from tasks has been updated. Starting with platform version `1.4.0`, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see [Fargate Task Networking](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- Task ENIs add support for jumbo frames. Network interfaces are configured with a maximum transmission unit (MTU), which is the size of the largest payload that fits within a single frame. The larger the MTU, the more application payload can fit within a single frame, which reduces per-frame overhead and increases efficiency. Supporting jumbo frames will reduce overhead when the network path between your task and the destination supports jumbo frames, such as all traffic that remains within your VPC.
- CloudWatch Container Insights will include network performance metrics for Fargate tasks. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 645\)](#).

- 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 \(p. 477\)](#).
- Added support for the `SYS_PTRACE` Linux parameter in container definitions. For more information, see [Linux parameters \(p. 232\)](#).
- 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 \(p. 289\)](#).
- Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service. For more information, [Task 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 \(p. 240\)](#), [Container dependency \(p. 236\)](#), and [Container timeouts \(p. 237\)](#).
- 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 \(p. 303\)](#).
- 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 \(p. 303\)](#).
- 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 \(p. 225\)](#).
- 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 \(p. 645\)](#).
- Beginning on December 3, 2019, the Fargate Spot capacity provider is supported. For more information, see [AWS Fargate capacity providers \(p. 180\)](#).
- 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 \(p. 173\)](#).

- Added support for private registry authentication using AWS Secrets Manager. For more information, see [Private registry authentication for tasks \(p. 300\)](#).

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 \(p. 173\)](#).

- Added support for the Amazon ECS task metadata endpoint. For more information, see [Amazon ECS task metadata endpoint \(p. 476\)](#).
- Added support for Docker health checks in container definitions. For more information, see [Health check \(p. 218\)](#).
- Added support for Amazon ECS service discovery. For more information, see [Service Discovery \(p. 599\)](#).

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 \(p. 173\)](#).

- 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](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.
- If you are using interface VPC endpoints, the following should be considered.
 - When using container images hosted with Amazon ECR, both the `com.amazonaws.region.ecr.dkr` and `com.amazonaws.region.ecr.api` Amazon ECR VPC endpoints as well as the Amazon S3 gateway endpoint are required. For more information, see [Amazon ECR interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon Elastic Container Registry User Guide*.
 - When using a task definition that references Secrets Manager secrets to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Secrets Manager. For more information, see [Using Secrets Manager with VPC Endpoints](#) in the *AWS Secrets Manager User Guide*.
 - When using a task definition that references Systems Manager Parameter Store parameters to retrieve sensitive data for your containers, you must create the interface VPC endpoints for Systems Manager. For more information, see [Using Systems Manager with VPC endpoints](#) in the *AWS Systems Manager User Guide*.

- Ensure that the security group in the Elastic Network Interface (ENI) associated with your task has the security group rules created to allow traffic between the task and the VPC endpoints you are using.

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 \(p. 172\)](#).

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 \(p. 169\)](#).

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 \(p. 559\)](#).
- 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

Getting started walkthroughs

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

- the section called “Getting started with the Amazon ECS console using Linux containers on AWS Fargate” (p. 30)
- the section called “Getting started with the Amazon ECS console using Windows containers on AWS Fargate” (p. 33)
- the section called “Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI” (p. 729)
- the section called “Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI” (p. 734)
- the section called “Tutorial: Creating a cluster with a Fargate task using the Amazon ECS CLI” (p. 62)

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 \(p. 178\)](#).

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 \(p. 175\)](#)
- [Creating a cluster \(p. 176\)](#)
- [Amazon ECS capacity providers \(p. 178\)](#)
- [Amazon ECS cluster auto scaling \(p. 189\)](#)
- [Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts \(p. 191\)](#)
- [Updating cluster settings \(p. 192\)](#)
- [Deleting a cluster \(p. 192\)](#)

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 \(p. 247\)](#).

- 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 \(p. 178\)](#).
- For tasks using the EC2 launch type, clusters can contain multiple different container instance types, but each container instance may only be registered to one cluster at a time.
- Custom IAM policies may be created to allow or restrict user access to specific clusters. For more information, see the [Cluster examples \(p. 668\)](#) section in [Amazon Elastic Container Service identity-based policy examples \(p. 663\)](#).

Creating a cluster

You can create an Amazon ECS cluster using the AWS Management Console, as described in this topic. Before you begin, be sure that you've completed the steps in [Setting up with Amazon ECS \(p. 7\)](#). You can register Amazon EC2 instances during cluster creation or register additional instances with the cluster after creating it.

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

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

To create a cluster (AWS Management 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, choose **Create Cluster**.
5. For **Select cluster compatibility**, choose one of the following options and then choose **Next Step**:
 - **Networking only**– This cluster template creates an empty cluster. Optionally, you can create a new VPC to use. This cluster template is typically used for workloads hosted on either AWS Fargate or external instances (ECS Anywhere). The **FARGATE** and **FARGATE_SPOT** capacity providers will be automatically associated with the cluster. For more information, see [AWS Fargate capacity providers \(p. 180\)](#).
 - **EC2 Linux + Networking**– This cluster template is used to create a cluster of Amazon EC2 instances to run Linux-based containers on. An Auto Scaling group is created for the Amazon EC2 instances.
 - **EC2 Windows + Networking** – This cluster template is used to create a cluster of Amazon EC2 instances to run Windows-based containers on. An Auto Scaling group is created for the Amazon EC2 instances. For more information, see [Amazon EC2 Windows containers \(p. 842\)](#).

Using the Networking only template

If you chose the **Networking only** cluster template, complete the following steps. Otherwise, skip to [Using the EC2 Linux + Networking or EC2 Windows + Networking template \(p. 177\)](#).

Using the Networking only cluster template

1. On the **Configure cluster** page, enter a **Cluster name**. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
2. In the **Networking** section, configure the VPC for your cluster. You can keep the default settings, or you can modify these settings with the following steps.
 - a. (Optional) If you choose to create a new VPC, for **CIDR Block**, select a CIDR block for your VPC. For more information, see [Your VPC and Subnets](#) in the *Amazon VPC User Guide*.
 - b. For **Subnets**, select the subnets to use for your VPC. You can keep the default settings, or you can modify them to meet your needs.
3. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see [Tagging Your Amazon ECS Resources](#).
4. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 645\)](#).
5. Choose **Create**.

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

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

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

1. For **Cluster name**, enter a name for your cluster. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
2. (Optional) To create a cluster with no resources, choose **Create an empty cluster**, **Create**.
3. For **Provisioning model**, choose one of the following instance types:
 - **On-Demand Instance**— With On-Demand Instances, you pay for compute capacity by the hour with no long-term commitments or upfront payments.
 - **Spot**— Spot Instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. For more information, see [Spot Instances](#).

Note

Spot Instances are subject to possible interruptions. We recommend that you avoid Spot Instances for applications that can't be interrupted. For more information, see [Spot Instance Interruptions](#).

4. For Spot Instances, do the following; otherwise, skip to the next step.
 - a. For **Spot Instance allocation strategy**, choose the strategy that meets your needs. For more information, see [Spot Fleet Allocation Strategy](#).
 - b. For **Maximum bid price (per instance/hour)**, specify a bid price. If your bid price is lower than the Spot price for the instance types that you selected, your Spot Instances are not launched.
5. For **EC2 instance type**, choose the Amazon EC2 instance type for your container instances. The instance type that you select determines the EC2 AMI IDs and resources available for your tasks. For GPU workloads, choose an instance type from the P2 or P3 instance family. For more information, see [Working with GPUs on Amazon ECS \(p. 250\)](#).
6. For **Number of instances**, choose the number of EC2 instances to launch into your cluster. These instances are launched using the latest Amazon ECS-optimized Amazon Linux AMI required by the instance type you chose. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).

7. For **EC2 AMI Id**, choose the Amazon ECS-optimized AMI for your container instances. The available AMIs will be determined by the Region and EC2 instance type you chose. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).
8. For **EBS storage (GiB)**, choose the size of the Amazon EBS volume to use for data storage on your container instances. You can increase the size of the data volume to allow for greater image and container storage.
9. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for SSH access. If you do not specify a key pair, you cannot connect to your container instances with SSH. For more information, see [Amazon EC2 Key Pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.
10. In the **Networking** section, configure the VPC to launch your container instances into. By default, the cluster creation wizard creates a new VPC with two subnets in different Availability Zones, and a security group open to the internet on port 80. This is a basic setup that works well for an HTTP service. However, you can modify these settings by following the substeps below.
 - a. For **VPC**, create a new VPC, or select an existing VPC.
 - b. (Optional) If you chose to create a new VPC, for **CIDR Block**, select a CIDR block for your VPC. For more information, see [Your VPC and Subnets](#) in the *Amazon VPC User Guide*.
 - c. For **Subnets**, select the subnets to use for your VPC. If you chose to create a new VPC, you can keep the default settings or you can modify them to meet your needs. If you chose to use an existing VPC, select one or more subnets in that VPC to use for your cluster.
 - d. For **Security group**, select the security group to attach to the container instances in your cluster. If you choose to create a new security group, you can specify a CIDR block to allow inbound traffic from. The default port 0.0.0.0/0 is open to the internet. You can also select a single port or a range of contiguous ports to open on the container instance. For more complicated security group rules, you can choose an existing security group that you have already created.

Note

You can also choose to create a new security group and then modify the rules after the cluster is created. For more information, see [Amazon EC2 Security Groups for Linux Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

- e. In the **Container instance IAM role** section, select the IAM role to use with your container instances. If your account has the **ecsInstanceRole** that is created for you in the console first-run wizard, it is selected by default. If you do not have this role in your account, you can choose to create the role, or you can choose another IAM role to use with your container instances.

Important

The IAM role you use must have the **AmazonEC2ContainerServiceforEC2Role** managed policy attached to it, otherwise you will receive an error during cluster creation. If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent does not connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

- f. If you chose the Spot Instance type earlier, the **Spot Fleet Role IAM role** section indicates that an IAM role **ecsSpotFleetRole** is created.
- g. In the **Tags** section, specify the key and value for each tag to associate with the cluster. For more information, see [Tagging Your Amazon ECS Resources](#).
- h. In the **CloudWatch Container Insights** section, choose whether to enable Container Insights for the cluster. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 645\)](#).
- i. 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 \(p. 180\)](#).

For Amazon ECS on Amazon EC2 users, a capacity provider consists of a capacity provider name, an Auto Scaling group, and the settings for managed scaling and managed termination protection. With managed scaling, Amazon ECS manage the scale-in and scale-out actions of the Auto Scaling group which provides auto scaling for your cluster's infrastructure. For more information, see [Auto Scaling group capacity providers \(p. 184\)](#).

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 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 managed termination protection is enabled, managed scaling must also be enabled 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 enable 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 \(p. 178\)](#).

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 support for Windows containers on Fargate.
- 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 \(p. 182\)](#).
- 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 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) or 1.0.0 or later (for Windows). For more information, see [AWS Fargate platform versions \(p. 169\)](#).
- 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 \(p. 636\)](#).

- 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 \(p. 237\)](#).
- 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": {  
        "clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",  
        "taskArn": "arn:aws:ecs:us-east-1:111122223333:task/  
b99d40b3-5176-4f71-9a52-9dbd6fEXAMPLE",  
        "status": "STOPPED",  
        "stopCode": "TerminationNotice",  
        "stoppedReason": "Your Spot Task was interrupted."  
    }  
}
```

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

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 \(p. 181\)](#).

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
```

Auto Scaling group capacity providers

Amazon ECS capacity providers can use Auto Scaling groups to manage the Amazon EC2 instances registered to their clusters. You can use the managed scaling feature to have Amazon ECS manage the scale-in and scale-out actions of the Auto Scaling group or you can manage the scaling actions yourself. For more information, see [Amazon ECS cluster auto scaling \(p. 189\)](#).

Topics

- [Auto Scaling group capacity providers considerations \(p. 184\)](#)
- [Creating an Auto Scaling group \(p. 184\)](#)
- [Creating an Auto Scaling group capacity provider \(p. 185\)](#)
- [Updating an Auto Scaling group capacity provider \(p. 186\)](#)
- [Creating a cluster with an Auto Scaling group capacity provider \(p. 187\)](#)
- [Deleting an Auto Scaling group capacity provider \(p. 188\)](#)

Auto Scaling group capacity providers considerations

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

- It is recommended that you create a new empty Auto Scaling group to use with a capacity provider rather than using an existing one. If you use an existing Auto Scaling group, any Amazon EC2 instances associated with the group that were already running and registered to an Amazon ECS cluster prior to the Auto Scaling group being used to create a capacity provider may not be properly registered with the capacity provider. This may cause issues when using the capacity provider in a capacity provider strategy. The `DescribeContainerInstances` API can confirm whether a container instance is associated with a capacity provider or not.

Note

To create an empty Auto Scaling group, set the desired count to zero. After you have created the capacity provider and associated it with a cluster, you can then scale it out.

- An Auto Scaling group must have a `MaxSize` greater than zero to enable it to scale out.
- If the Auto Scaling group is unable to scale out to accommodate the number of tasks run, the tasks will fail to transition beyond the `PROVISIONING` state.
- When using managed termination protection, managed scaling must be enabled otherwise managed termination protection will not work.
- When using managed scaling, the Auto Scaling group shouldn't have any scaling policies attached to it other than the ones Amazon ECS creates, otherwise the Amazon ECS created scaling plans will receive an `ActiveWithProblems` error. For more information, see [Avoiding the ActiveWithProblems error](#) in the [AWS Auto Scaling User Guide](#).

Creating an Auto Scaling group

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

Note

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

For more information on replacing an Auto Scaling launch configuration with an Amazon EC2 launch template, see [Replacing a launch configuration with a launch template](#) in the *Amazon EC2 Auto Scaling User Guide*

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

- If managed termination protection is enabled when you create a capacity provider, the Auto Scaling group and each Amazon EC2 instance in the Auto Scaling group must have instance protection from scale in enabled as well. For more information, see [Instance Protection](#) in the *AWS Auto Scaling User Guide*.
- If managed scaling is enabled when you create a capacity provider, the Auto Scaling group desired count can be set to 0. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group.

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

For more information on creating an Amazon EC2 Auto Scaling group, see [Auto Scaling groups](#) in the *Amazon EC2 Auto Scaling User Guide*.

Creating an Auto Scaling group capacity provider

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

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

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

To create an Auto Scaling group capacity provider (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region your cluster exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose **Capacity Providers**, and then choose **Create**.
6. For **Capacity provider name**, enter a capacity provider name.
7. For **Auto Scaling group**, select the Auto Scaling group to associate with the capacity provider. The Auto Scaling group must already be created. For more information, see [Creating an Auto Scaling group \(p. 184\)](#).
8. For **Managed scaling**, choose your managed scaling option. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling plans. When managed scaling is disabled, you manage your Auto Scaling groups yourself.
9. For **Target capacity %**, if managed scaling is enabled, specify an integer between 1 and 100. The target capacity value is used as the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy. This target capacity value is matched on a best effort

basis. For example, a value of 100 will result in the Amazon EC2 instances in your Auto Scaling group being completely utilized and any instances not running any tasks will be scaled in, but this behavior is not guaranteed at all times.

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

To create an Auto Scaling group capacity provider (AWS CLI)

- Use the following command to create a new capacity provider.
- [create-capacity-provider \(AWS CLI\)](#)

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

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

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

Updating an Auto Scaling group capacity provider

A capacity provider can be updated to change its managed scaling and managed termination protection settings. Use the following steps to update an existing capacity provider.

To update an Auto Scaling group capacity provider (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region the cluster the capacity provider is associated with exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose the **Capacity Providers** tab.
6. Select the capacity provider to update and choose **Update**.
7. On the **Update Capacity Provider** page, the following parameters can be updated.
 - a. For **Managed scaling**, choose your managed scaling option. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling plans. When managed scaling is disabled, you manage your Auto Scaling groups yourself.
 - b. For **Target capacity %**, if managed scaling is enabled, specify an integer between 1 and 100. The target capacity value is used as the target value for the CloudWatch metric used in the

Amazon ECS-managed target tracking scaling policy. This target capacity value is matched on a best effort basis. For example, a value of 100 will result in the Amazon EC2 instances in your Auto Scaling group being completely utilized and any instances not running any tasks will be scaled in, but this behavior is not guaranteed at all times.

- c. For **Managed termination protection**, choose your managed termination protection option. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled and if managed scaling is enabled. Managed termination protection is only supported on standalone tasks or tasks in a service using the replica scheduling strategy. For tasks in a service using the daemon scheduling strategy, the instances are not protected.
8. Choose **Update** to request capacity provider update.
9. To verify whether the capacity provider update was successful, check the **Update Status** column on the **Capacity Providers** tab.

To update an Auto Scaling group capacity provider (AWS CLI)

- Use the following command to create a new capacity provider.
 - [update-capacity-provider \(AWS CLI\)](#)

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

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

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

Creating a cluster with an Auto Scaling group capacity provider

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

For AWS Management Console steps, see [Creating a cluster \(p. 176\)](#).

To create a cluster with an Auto Scaling group capacity provider (AWS CLI)

Use the following command to create a new cluster and associate one or more capacity providers with it.

- [create-cluster \(AWS CLI\)](#)

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

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

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

Deleting an Auto Scaling group capacity provider

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

Prior to an Auto Scaling group capacity provider being deleted, the capacity provider must be removed from the capacity provider strategy from all services. The `UpdateService` API or the update service workflow in the AWS Management Console can be used to remove a capacity provider from a service's capacity provider strategy. The force new deployment option can be used to ensure that any tasks using the Amazon EC2 instance capacity provided by the capacity provider are transitioned to use the capacity from the remaining capacity providers.

There are other prerequisites that must be performed to delete a capacity provider but they are specific to the tool used and are mentioned in the following steps.

Use the following steps to delete an Auto Scaling group capacity provider.

To delete an Auto Scaling group capacity provider (AWS Management Console)

When deleting a capacity provider using the AWS Management Console, the console goes through two steps. The capacity provider is first disassociated from the cluster completely and then it is deleted. In rare cases, the capacity provider may be successfully disassociated from the cluster but is unable to be deleted. In those cases, you must use either the Amazon ECS API or the AWS CLI to view the status of the capacity provider and delete it.

Note

Only capacity providers that are currently associated with a cluster are visible in the AWS Management Console. To delete a capacity provider that is not associated with a cluster, you must use the Amazon ECS API, SDK, or AWS CLI.

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region your cluster exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose the **Capacity Providers** tab.
6. Select the capacity provider you want to delete and then choose **Delete**.

To delete an Auto Scaling group capacity provider (AWS CLI)

When using the AWS CLI to delete a capacity provider, the capacity provider must first be disassociated from the cluster. The following options are available to disassociate a capacity provider from a cluster.

Option 1: Use the `delete` command to delete the cluster. This will disassociate the capacity provider from the cluster upon successful deletion of the cluster.

- [delete-cluster](#) (AWS CLI)

```
aws ecs delete-cluster \
--cluster MyCluster
```

Option 2: Use the **put-cluster-capacity-providers** command to disassociate a capacity provider from a cluster. If you have other capacity providers associated with the cluster that you want to have remain associated with the cluster, you must include those when using the command.

The following example will remove all existing capacity providers from the specified cluster.

- [put-cluster-capacity-providers \(AWS CLI\)](#)

```
aws ecs put-cluster-capacity-providers \
--cluster MyCluster \
--capacity-providers [] \
--default-capacity-provider-strategy []
```

Use the **delete-capacity-provider** command to delete a capacity provider. You can specify the capacity provider using its short name or the full Amazon Resource Name (ARN).

- [delete-capacity-provider \(AWS CLI\)](#)

Example using the short name:

```
aws ecs delete-capacity-provider \
--capacity-provider ExampleCapacityProvider
```

Example using the full ARN:

```
aws ecs delete-capacity-provider \
--capacity-provider arn:aws:ecs:us-west-2:123456789012:capacity-
provider/ExampleCapacityProvider
```

Amazon ECS cluster auto scaling

Amazon ECS cluster auto scaling enables you to have more control over how you scale the Amazon EC2 instances within a cluster. When creating an Auto Scaling group capacity provider with managed scaling enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group used when creating the capacity provider. On your behalf, Amazon ECS creates an AWS Auto Scaling scaling plan with a target tracking scaling policy based on the target capacity value you specify. Amazon ECS then associates this scaling plan with your Auto Scaling group.

For each of the Auto Scaling group capacity providers with managed scaling enabled, an Amazon ECS managed CloudWatch metric with the prefix `AWS/ECS/ManagedScaling` is created along with two CloudWatch alarms. The CloudWatch metrics and alarms are used to monitor the Amazon EC2 instance capacity in your Auto Scaling groups and will trigger the Auto Scaling group to scale in and scale out as needed.

Each cluster has one or more Auto Scaling group capacity providers and an optional default capacity provider strategy. The capacity providers determine the infrastructure to use for the tasks, and the capacity provider strategy determines how the tasks are spread across the capacity providers. When you run a task or create a service, you may either use the cluster's default capacity provider strategy or specify a capacity provider strategy that overrides the cluster's default strategy. For more information about capacity providers, see [Amazon ECS capacity providers \(p. 178\)](#).

Cluster auto scaling considerations

The following should be considered when using cluster auto scaling:

- Amazon ECS uses the `AWSServiceRoleForECS` service-linked IAM role for the permissions it requires to call AWS Auto Scaling, on your behalf. For more information on using and creating Amazon ECS service-linked IAM roles, see [Service-linked role for Amazon ECS \(p. 685\)](#).
- Cluster auto scaling is not available in the Asia Pacific (Osaka) Region.
- When using capacity providers with Auto Scaling groups, the IAM user creating the capacity providers, needs the `autoscaling:CreateOrUpdateTags` permission. This is because Amazon ECS adds a tag to the Auto Scaling group when it associates it with the capacity provider.

Important

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

- Managed scaling works best if your Auto Scaling group uses the same or similar instance types. For more information, see [Managed scale-out behavior \(p. 190\)](#).
- When using an Auto Scaling group with On-Demand instances and multiple instance types, place the larger instance types higher in the priority list and don't specify a weight. Specifying a weight is not supported at this time. For more information, see [Auto Scaling groups with multiple instance types](#) in the [AWS Auto Scaling User Guide](#).
- When creating a service, specifying a task placement strategy that spreads across Availability Zones or a binpack strategy based on CPU or memory works best. Don't use an instance spread strategy as scaling works slower with that strategy type.
- The desired capacity for the Auto Scaling group associated with a capacity provider shouldn't be changed or managed by any scaling policies other than the one Amazon ECS manages.

Managed scale-out behavior

When using Auto Scaling group capacity providers with managed scaling enabled, Amazon ECS estimates the lower bound on the optimal number of instances to add to your cluster and uses this value to determine how many instances to request. The following describes the scale-out behavior in more detail.

1. Group all of the provisioning tasks so that each group has the same exact resource requirements.
2. When multiple instance types are used, the instances in the Auto Scaling group are sorted by their attributes, such as vCPU, memory, elastic network interface (ENI), ports, and GPUs and the largest instance types for each attribute are selected.
3. For each group of tasks, the number of instances required to run the unplaced tasks is calculated. This calculation uses a binpack strategy which accounts for the vCPU, memory, elastic network interfaces (ENI), ports, and GPUs requirements of the tasks and the resource availability of the Amazon EC2 instances. This value will be treated as the maximum calculated instance count.

Note

This calculation takes into account any task placement constraints that are defined, but we recommend only using the `distinctInstance` task placement constraint.

4. Amazon ECS will then launch either the `minimumScalingStepSize`, if the maximum calculated instance count is less than the minimum scaling step size, or the lower of either the `maximumScalingStepSize` or the maximum calculated instance count value.

For a more detailed explanation of how this logic works, see [Deep dive on Amazon ECS cluster auto scaling](#).

Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts

Amazon ECS supports workloads that take advantage of Local Zones, Wavelength Zones and AWS Outposts when low latency or local data processing requirements are needed.

- Local Zones are an extension of an AWS Region that provide you the ability to place resources in multiple locations closer to your end users.
- Wavelength Zones allow developers to build applications that deliver ultra-low latencies to 5G devices and end users. Wavelength deploys standard AWS compute and storage services to the edge of telecommunication carriers' 5G networks.
- AWS Outposts brings native AWS services, infrastructure, and operating models to virtually any data center, co-location space, or on-premises facility.

Important

Amazon ECS on AWS Fargate workloads are not supported in Local Zones, Wavelength Zones, or on AWS Outposts at this time.

We describe each of these in more detail in the following section.

Local Zones

A *Local Zone* is an extension of an AWS Region in geographic proximity to your users. Local Zones have their own connections to the internet and support AWS Direct Connect, so resources created in a Local Zone can serve local users with low-latency communications. For more information, see [AWS Local Zones](#).

A Local Zone is represented by a Region code followed by an identifier that indicates the location, for example, `us-west-2-lax-1a`.

To use a Local Zone, you must opt-in to the zone. Once you have opted in, you must create a Amazon VPC and subnet in the Local Zone. Then you will be ready to launch your Amazon EC2 instances, Amazon FSx file servers, and Application Load Balancers in them to use for your Amazon ECS clusters and tasks. For more information, see [Local Zones](#) in the *Amazon EC2 User Guide for Linux Instances*.

Wavelength Zones

AWS Wavelength allows developers to build applications that deliver ultra-low latencies to mobile devices and end users. Wavelength deploys standard AWS compute and storage services to the edge of telecommunication carriers' 5G networks. Developers can extend a Amazon Virtual Private Cloud to one or more Wavelength Zones, and then use AWS resources like Amazon EC2 instances to run applications that require ultra-low latency and a connection to AWS services in the Region.

A Wavelength Zone is an isolated zone in the carrier location where the Wavelength infrastructure is deployed. Wavelength Zones are tied to a Region. A Wavelength Zone is a logical extension of a Region, and is managed by the control plane in the Region.

A Wavelength Zone is represented by a Region code followed by an identifier that indicates the Wavelength Zone, for example, `us-east-1-wl1-bos-wlz-1`.

To use a Wavelength Zone, you must opt-in to the zone. Once you have opted in, you must create a Amazon VPC and subnet in the Wavelength Zone. Then you will be ready to launch your Amazon EC2 instances in them to use for your Amazon ECS clusters and tasks. For more information, see [Get started with AWS Wavelength](#) in the *AWS Wavelength Developer Guide*.

Wavelength Zones are not available in every Region. For information about the Regions that support Wavelength Zones, see [Available Wavelength Zones in the AWS Wavelength Developer Guide](#).

AWS Outposts

AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. Amazon ECS on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see [Amazon Elastic Container Service on AWS Outposts \(p. 722\)](#).

Updating cluster settings

Cluster settings enable you to configure parameters for your existing Amazon ECS clusters. You can update cluster settings using the Amazon ECS API, AWS CLI or SDKs. Currently, the only supported cluster setting is `containerInsights`, which allows you to enable or disable CloudWatch Container Insights for an existing cluster. To enable CloudWatch Container Insights for a new cluster, that can be done in the AWS Management Console during cluster creation. For more information, see [Creating a cluster \(p. 176\)](#).

Important

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

To update the settings for a cluster (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

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

When you delete a cluster in the Amazon ECS console, the associated resources that are deleted will 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*.

Important

The delete cluster workflow is not supported yet in the new Amazon ECS console.

To delete a cluster (AWS Management 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**.

Important

In the navigation pane, turn off **New ECS Experience**, or choose **use the old console**. The delete cluster workflow is not supported yet in the new Amazon ECS console.

4. On the **Clusters** page, select the cluster to delete.

Note

If your cluster has registered container instances, you must deregister or terminate them.

For more information, see [Deregister an Amazon EC2 backed container instance \(p. 429\)](#).

5. In the upper-right of the page, choose **Delete Cluster**. You see one of two confirmation prompts:

- **Deleting the cluster also deletes the AWS CloudFormation stack**

EC2ContainerService-*cluster_name* – Deleting this cluster cleans up the associated resources that were created with the cluster, including Auto Scaling groups, VPCs, or load balancers.

- **Deleting the cluster does not affect AWS CloudFormation resources** – Deleting this cluster does not clean up any resources that are associated with the cluster, including Auto Scaling groups, VPCs, or load balancers. Also, any container instances that are registered with this cluster must be deregistered or terminated before you can delete the cluster. For more information, see [Deregister an Amazon EC2 backed container instance \(p. 429\)](#). You can visit the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation/> to update or delete any of these resources.

6. In the confirmation box, enter **delete me**.

Amazon ECS task definitions

A task definition is required to run Docker containers in Amazon ECS. The following are some of the parameters 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 or each container within a task
- The launch type to use, which determines the infrastructure on which your tasks are hosted
- The Docker networking mode to use for the containers in your task
- The logging configuration to use for your tasks
- Whether the task should continue to run if the container finishes or fails
- The command the container should run when it is started
- Any data volumes that should be used with the containers in the task
- The IAM role that your tasks should use

You can define multiple containers in a task definition. The parameters that you use depend on the launch type you choose for the task. Not all parameters are valid. For more information about the parameters available and which launch types they are valid for in a task definition, see [Task definition parameters \(p. 209\)](#).

Your entire application stack does not need to be on a single task definition, and in most cases it should not. 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 \(p. 195\)](#).

Topics

- [Application architecture \(p. 195\)](#)
- [Creating a task definition using the new console \(p. 196\)](#)
- [Creating a task definition using the classic console \(p. 198\)](#)
- [Task definition parameters \(p. 209\)](#)
- [Amazon ECS launch types \(p. 247\)](#)
- [Working with GPUs on Amazon ECS \(p. 250\)](#)
- [Working with inference workloads on Amazon ECS \(p. 253\)](#)
- [Using data volumes in tasks \(p. 256\)](#)
- [Managing container swap space \(p. 277\)](#)
- [Amazon ECS task networking \(p. 278\)](#)
- [Using the awslogs log driver \(p. 283\)](#)
- [Custom log routing \(p. 289\)](#)
- [Private registry authentication for tasks \(p. 300\)](#)
- [Specifying sensitive data \(p. 303\)](#)
- [Specifying environment variables \(p. 315\)](#)
- [Example task definitions \(p. 317\)](#)

- [Updating a task definition \(p. 322\)](#)
- [Deregistering a task definition revision \(p. 323\)](#)

Application architecture

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

Using the Fargate launch type

When architecting your application to run on Amazon ECS using AWS Fargate, the main question is when should you put multiple containers into the same task definition versus deploying containers separately in multiple task definitions.

When the following conditions are required, we recommend that you deploy your containers in a single task definition:

- Your containers share a common lifecycle (that is, they are 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.

Otherwise, you should define your containers in separate tasks definitions so that you can scale, provision, and deprovision them separately.

Using the EC2 launch type

When you're considering how to model task definitions and services using the EC2 launch type, it helps to think about what processes need to run together and how to scale each component.

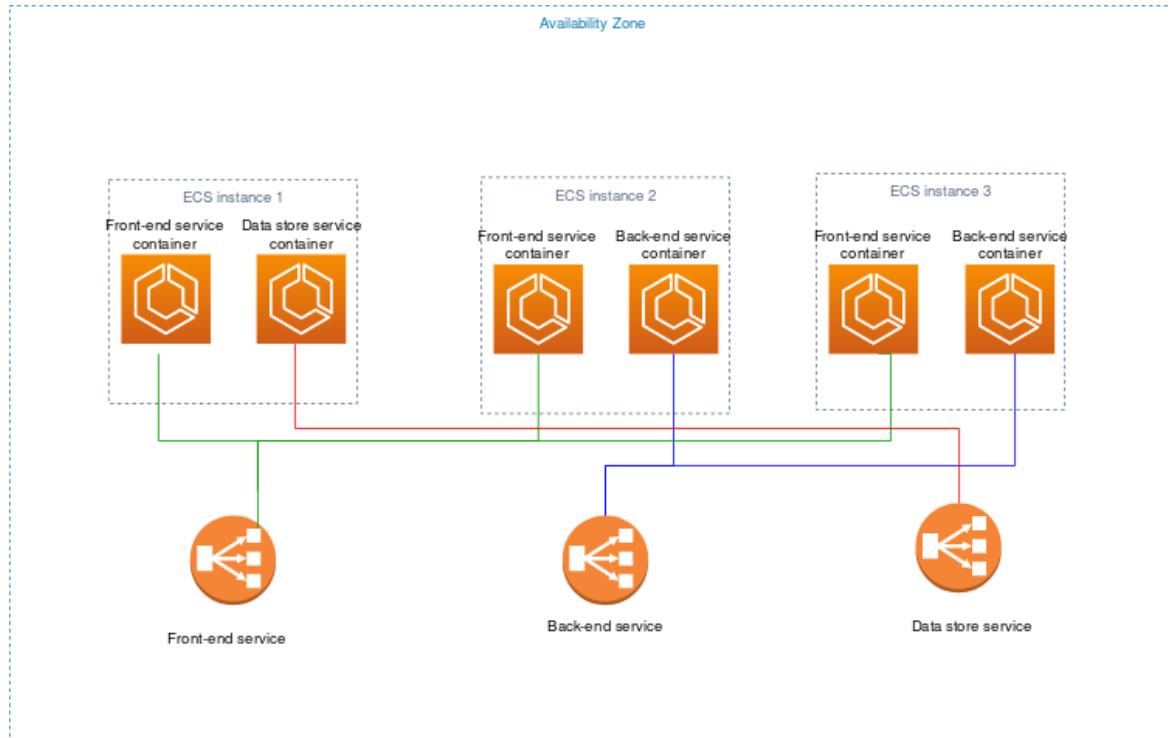
As an example, imagine an application that consists of the following components:

- A frontend service that displays information on a webpage
- A backend service that provides APIs for the frontend service
- A data store

In your development environment, you probably run all three of these containers together on your Docker host. You might be tempted to use the same approach for your production environment, but this approach has several drawbacks:

- Changes to one component can impact all three of the components, which may be a larger scope for the change than anticipated.
- Each component is more difficult to scale because you have to scale every container proportionally.
- Task definitions can only have 10 container definitions, but your application stack might require more definitions, either now or in the future.
- Every container in a task definition must land on the same container instance, which can limit your instance choices to the largest sizes.

Given these drawbacks, you should create task definitions that group the containers that are used for a common purpose, and separate the different components into multiple, separate task definitions. In this preceding example, three task definitions each specify one container. The following example cluster (illustrated in the figure below) has three container instances registered with three front-end service containers, two back-end service containers, and one data store service container.



You can group related containers in a task definition, such as linked containers that must be run together. For example, you could add a log streaming container to your front-end service and include it in the same task definition.

After you have your task definitions, you can create services from them to maintain the availability of your desired tasks. For more information, see [Creating an Amazon ECS service \(p. 546\)](#). In your services, you can associate containers with Elastic Load Balancing load balancers. For more information, see [Service load balancing \(p. 574\)](#). When your application requirements change, you can update your services to scale the number of desired tasks up or down, or to deploy newer versions of the containers in your tasks. For more information, see [Updating a service \(p. 559\)](#).

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 Amazon ECS 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 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 Docker Hub registry are available by default. You can also specify other repositories using either the `repository-url/image:tag` or `repository-url/image@digest` formats.
 - c. For **Container port** and **Protocol**, specify the port mapping to use for the container. A port mapping allows the container to access ports on the host to send or receive traffic.
 - d. 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.
 - 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 ECS performs validation using this value to ensure the task definition parameters are valid for the infrastructure type.
 6. 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 MB, 1 GB, 2 GB
512 (.5 vCPU)	1 GB, 2 GB, 3 GB, 4 GB
1024 (1 vCPU)	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2048 (2 vCPU)	Between 4 GB and 16 GB in 1 GB increments
4096 (4 vCPU)	Between 8 GB and 30 GB in 1 GB increments

For tasks hosted on Amazon EC2, supported task CPU values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

7. (Optional) Expand the **Task roles, network mode** section to specify an IAM role to assign to the task.
8. (Optional) The **Storage** section is used to expand the amount of ephemeral storage for tasks hosted on Fargate as well as the 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.
9. (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.
10. (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 \(p. 289\)](#).
- 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 \(p. 298\)](#).
 - **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.
11. (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.

Important

Your application must be setup to send trace data, otherwise no data is sent to AWS X-Ray.
For more information, see [Instrumenting your application for AWS X-Ray](#) in the [AWS X-Ray Developer Guide](#).

12. (Optional) Expand the **Tags** section to add tags, as key-value pairs, to the task definition.
13. Choose **Next** to review the task definition.
14. 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.

Creating a task definition using the classic console

Important

Amazon ECS has provided a new console experience for creating a task definition. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

Before you can run Docker containers on Amazon ECS, you must create a task definition. You can define multiple containers and data volumes in a single task definition. For more information about the parameters available in a task definition, see [Task definition parameters \(p. 209\)](#).

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 compatibilities** page, select the launch type that your task should use and choose **Next step**.
4. Follow the steps under one of the following tabs, according to the launch type that you have chosen.

Fargate launch type

Using the Fargate launch type compatibility template

If you chose **Fargate**, complete the following steps:

1. (Optional) If you have a JSON representation of your task definition, complete the following steps:

- a. On the **Configure task and container definitions** page, scroll to the bottom of the page and choose **Configure via JSON**.
- b. Paste your task definition JSON into the text area and choose **Save**.
- c. Verify your information and choose **Create**.

Scroll to the bottom of the page and choose **Configure via JSON**.

2. For **Task Definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
3. (Optional) For **Task Role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS API operations on your behalf. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

Note

Only roles that have the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For more information about creating an IAM role for your tasks, see [Creating an IAM Role and Policy for your Tasks \(p. 702\)](#).

4. For **Operating system family**, choose the container operating system.
5. 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 \(p. 691\)](#).
6. 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 MB, 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

7. For each container in your task definition, complete the following steps:
 - a. Choose **Add container**.
 - b. Fill out each required field and any optional fields to use in your container definitions. More container definition parameters are available in the **Advanced container configuration** menu. For more information, see [Task definition parameters \(p. 209\)](#).
 - c. Choose **Add** to add your container to the task definition.

8. (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.
9. (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](#).
10. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:
 - a. For **Type**, choose the log router type to use.
 - b. For **Image**, type the image URI for your log router container. If you chose the fluentbit log router type, the **Image** field prepopulates with the AWS for Fluent Bit image. For more information, see [Using the AWS for Fluent Bit image \(p. 291\)](#).
 - 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.
11. (Optional) To define data volumes for your task, choose **Add volume**. For more information, see [Using data volumes in tasks \(p. 256\)](#).
 - For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

12. 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](#).
13. Choose **Create**.

EC2 launch type

Using the EC2 launch type compatibility template

If you chose **EC2**, complete the following steps:

1. (Optional) If you have a JSON representation of your task definition, complete the following steps:
 - a. On the **Configure task and container definitions** page, scroll to the bottom of the page and choose **Configure via JSON**.
 - b. Paste your task definition JSON into the text area and choose **Save**.
 - c. Verify your information and choose **Create**.

Scroll to the bottom of the page and choose **Configure via JSON**.

2. For **task definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
3. (Optional) For **Task Role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

For tasks that use the EC2 launch type, these permissions are usually granted by the Amazon ECS Container Instance IAM role. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

Note

Only roles that have the **Amazon EC2 Container Service Task Role** trust relationship are shown here. For more information about creating an IAM role for your tasks, see [Creating an IAM Role and Policy for your Tasks \(p. 702\)](#).

4. (Optional) For **Network Mode**, choose the Docker network mode to use for the containers in your task. The available network modes correspond to those described in [Network settings](#) in the Docker run reference. If you select **Enable App Mesh integration** in a following step, then you must select `awsvpc`.

The default Docker network mode is `bridge`. If the network mode is set to `none`, you can't specify port mappings in your container definitions, and the task's containers don't have external connectivity. If the network mode is `awsvpc`, the task is allocated an elastic network interface. The `host` and `awsvpc` network modes offer the highest networking performance for containers. This is because they use the Amazon EC2 network stack instead of the virtualized network stack provided by the `bridge` mode. However, exposed container ports are mapped directly to the corresponding host port. Therefore, you cannot take advantage of dynamic host port mappings or run multiple instantiations of the same task on a single container instance if port mappings are used.

5. (Optional) For **Task execution role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf.

For tasks that use the EC2 launch type, these permissions are usually granted by the Amazon ECS Container Instance IAM role, which is specified earlier as the **Task Role**. There is no need to specify a task execution role. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

6. (Optional) For **Task size**, choose a value for **Task memory (GB)** and **Task CPU (vCPU)**. Supported Task CPU (vCPU) values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

Note

Task-level CPU and memory parameters are ignored for Windows containers. We recommend specifying container-level resources for Windows containers.

7. For each container in your task definition, complete the following steps.
 - a. Choose **Add container**.
 - b. Enter each of the 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 \(p. 209\)](#).
 - c. Choose **Add** to add your container to the task definition.
8. (Optional) For **Constraint**, you define how tasks that are created from this task definition are placed in your cluster. For tasks that use the EC2 launch type, you can use constraints to place tasks based on Availability Zone, instance type, or custom attributes. For more information, see [Amazon ECS task placement constraints \(p. 516\)](#).
9. (Optional) For **Service Integration**, to configure the parameters for App Mesh integration, choose **Enable App Mesh integration** and then do the following:
 - a. For **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](#).
 - 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.

10. (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](#).
11. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:
 - a. For **Type**, choose the log router type to use.
 - b. For **Image**, type the image URI for your log router container. If you chose the fluentbit log router type, the **Image** field prepopulates with the AWS for Fluent Bit image. For more information, see [Using the AWS for Fluent Bit image \(p. 291\)](#).
 - c. Choose **Apply**. This creates a new log router container to the task definition named `log_router`, and applies the settings to support it. If you make changes to the log router integration fields, choose **Apply** again to update the FireLens container.
12. (Optional) To define data volumes for your task, choose **Add volume**. You can create either a bind mount or Docker volume. For more information, see [Using data volumes in tasks \(p. 256\)](#).
 - a. For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - b. (Optional) To create a bind mount volume, for **Source path**, type the path on the host container instance to present to the container. If you leave this field empty, the Docker daemon assigns a host path for you. If you specify a source path, the data volume persists at the specified location on the host container instance until you delete it manually. If the source path doesn't exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported to the container.
 - c. To create a Docker volume, select **Specify a volume driver**.
 - i. For **Driver**, choose the Docker volume driver to use. The driver value must match the driver name provided by Docker. Use `docker plugin ls` on your container instance to retrieve the driver name.
 - ii. For **Scope**, choose the option that determines the lifecycle of the Docker volume. Docker volumes that are scoped to a `task` are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as `shared` persist after the task stops.
 - iii. Select **Enable auto-provisioning** to have the Docker volume created if it does not already exist. This option is only available for volumes that specify the `shared` scope.
 - iv. For **Driver options**, specify the driver-specific key values to use.
 - v. For **Volume labels**, specify the custom metadata to add to your Docker volume.
13. In the **Tags** section, specify the key and value for each tag to associate with the task definition. For more information, see [Tagging Your Amazon ECS Resources](#).
14. Choose **Create**.

External instance launch type

Using the external instance launch type

If you chose **External**, complete the following steps:

1. (Optional) If you have a JSON representation of your task definition, complete the following steps:
 - a. On the **Configure task and container definitions** page, scroll to the bottom of the page and choose **Configure via JSON**.
 - b. Paste your task definition JSON into the text area and choose **Save**.
 - c. Verify your information and choose **Create**.

Scroll to the bottom of the page and choose **Configure via JSON**.

2. For **task definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
3. (Optional) For **Task Role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf. For more information, see [IAM Roles for Tasks \(p. 699\)](#) and the section called “[IAM permissions](#)” (p. 416).
4. (Optional) For **Network Mode**, choose the Docker network mode to use for the containers in your task. The available network modes correspond to those described in [Network settings](#) in the Docker run reference.

The default Docker network mode is `bridge`. If the network mode is set to `none`, you can't specify port mappings in your container definitions, and the task's containers don't have external connectivity. If the network mode is `awsvpc`, the task is allocated an elastic network interface. The `host` and `awsvpc` network modes offer the highest networking performance for containers. This is because they use the Amazon EC2 network stack instead of the virtualized network stack provided by the `bridge` mode. However, exposed container ports are mapped directly to the corresponding host port. Therefore, you cannot take advantage of dynamic host port mappings or run multiple instantiations of the same task on a single container instance if port mappings are used.

5. (Optional) For **Task execution role**, choose an IAM role that provides permissions for containers in your task to make calls to AWS APIs on your behalf.
6. (Optional) For **Task size**, choose a value for **Task memory (GB)** and **Task CPU (vCPU)**. Supported `Task CPU (vCPU)` values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

Note

Task-level CPU and memory parameters are ignored for Windows containers. We recommend specifying container-level resources for Windows containers.

7. For each container in your task definition, complete the following steps.
 - a. Choose **Add container**.
 - b. Enter each of the 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 \(p. 209\)](#).
 - c. Choose **Add** to add your container to the task definition.
8. (Optional) For **Constraint**, you define how tasks that are created from this task definition are placed in your cluster. For more information, see [Amazon ECS task placement constraints \(p. 516\)](#).
9. (Optional) For **Log Router Integration**, you can add a custom log routing configuration. Choose **Enable FireLens integration** and then do the following:
 - a. For **Type**, choose the log router type to use.
 - b. For **Image**, type the image URI for your log router container. If you chose the `fluentbit` log router type, the **Image** field prepopulates with the AWS for Fluent Bit image. For more information, see [Using the AWS for Fluent Bit image \(p. 291\)](#).
 - c. Choose **Apply**. This creates a new log router container to the task definition named `log_router`, and applies the settings to support it. If you make changes to the log router integration fields, choose **Apply** again to update the FireLens container.
10. (Optional) To define data volumes for your task, choose **Add volume**. You can create either a bind mount or Docker volume. For more information, see [Using data volumes in tasks \(p. 256\)](#).
 - a. For **Name**, type a name for your volume. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

- b. (Optional) To create a bind mount volume, for **Source path**, type the path on the host container instance to present to the container. If you leave this field empty, the Docker daemon assigns a host path for you. If you specify a source path, the data volume persists at the specified location on the host container instance until you delete it manually. If the source path doesn't exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported to the container.
 - c. To create a Docker volume, select **Specify a volume driver**.
 - i. For **Driver**, choose the Docker volume driver to use. The driver value must match the driver name provided by Docker. Use `docker plugin ls` on your container instance to retrieve the driver name.
 - ii. For **Scope**, choose the option that determines the lifecycle of the Docker volume. Docker volumes that are scoped to a task are automatically provisioned when the task starts and destroyed when the task stops. Docker volumes that are scoped as shared persist after the task stops.
 - iii. Select **Enable auto-provisioning** to have the Docker volume created if it does not already exist. This option is only available for volumes that specify the shared scope.
 - iv. For **Driver options**, specify the driver-specific key values to use.
 - v. For **Volume labels**, specify the custom metadata to add to your Docker volume.
11. In the **Tags** section, specify the key and value for each tag to associate with the task definition. For more information, see [Tagging Your Amazon ECS Resources](#).
12. Choose **Create**.

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 \(p. 209\)](#).

```
{
  "family": "",
  "taskRoleArn": "",
  "executionRoleArn": "",
  "networkMode": "bridge",
  "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": "START"
        }
    ],
    "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": "memlock",
            "softLimit": 0,
            "hardLimit": 0
        }
    ],
    "logConfiguration": {
        "logDriver": "splunk",
        "options": {
            "KeyName": ""
        }
    },
    "secretOptions": [
        {
            "name": "",
            "valueFrom": ""
        }
    ]
},
"healthCheck": {
    "command": [
        ""
    ],
    "interval": 0,
    "timeout": 0,
    "retries": 0,
    "startPeriod": 0
},
"systemControls": [
{
```

```

        "namespace": "",
        "value": ""
    }
],
"resourceRequirements": [
{
    "value": "",
    "type": "InferenceAccelerator"
}
],
"firelensConfiguration": {
    "type": "fluentbit",
    "options": {
        "KeyName": ""
    }
}
],
"volumes": [
{
    "name": "",
    "host": {
        "sourcePath": ""
    },
    "dockerVolumeConfiguration": {
        "scope": "shared",
        "autop provision": true,
        "driver": "",
        "driverOpts": {
            "KeyName": ""
        },
        "labels": {
            "KeyName": ""
        }
    },
    "efsVolumeConfiguration": {
        "fileSystemId": "",
        "rootDirectory": "",
        "transitEncryption": "DISABLED",
        "transitEncryptionPort": 0,
        "authorizationConfig": {
            "accessPointId": "",
            "iam": "DISABLED"
        }
    }
}
],
"placementConstraints": [
{
    "type": "memberOf",
    "expression": ""
}
],
"requiresCompatibilities": [
    "FARGATE"
],
"cpu": "",
"memory": "",
"tags": [
{
    "key": "",
    "value": ""
}
],
"pidMode": "host",
"ipcMode": "none",

```

```
"proxyConfiguration": {  
    "type": "APPmesh",  
    "containerName": "",  
    "properties": [  
        {  
            "name": "",  
            "value": ""  
        }  
    ]  
},  
"inferenceAccelerators": [  
    {  
        "deviceName": "",  
        "deviceType": ""  
    }  
]
```

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

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

Task definition parameters

Task definitions are split into separate parts: the task family, the IAM task role, the network mode, container definitions, volumes, task placement constraints, and launch types. The family and container definitions are required in a task definition, while task role, network mode, volumes, task placement constraints, and launch type are optional.

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

Family

`family`

Type: string

Required: yes

When you register a task definition, you give it a family, which is similar to a name for multiple versions of the task definition, specified with a revision number. The first task definition that is registered into a particular family is given a revision of 1, and any task definitions registered after that are given a sequential revision number.

Platform family

`platformFamily`

Type: string

Required: Conditional

Default: LINUX

The operating system of the instance that runs the Amazon ECS service;

The valid values are `LINUX`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`.

This is required for Fargate tasks.

All tasks that run as part of this service must use the same `platformFamily` value as the service, for example, `LINUX`.

Launch types

When you register a task definition, you can specify a launch type that Amazon ECS should validate the task definition against. A client exception is returned if the task definition doesn't validate against the compatibilities specified. For more information, see [Amazon ECS launch types \(p. 247\)](#).

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 used in the task definition meet the requirements of the launch type.

Task role

`taskRoleArn`

Type: string

Required: no

When you register a task definition, you can provide a task role for an IAM role that allows the containers in the task permission to call the AWS APIs that are specified in its associated policies on your behalf. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

IAM roles for tasks on Windows require that the `-EnableTaskIAMRole` option is set when you launch the Amazon ECS-optimized Windows Server AMI. Your containers must also run some configuration code in order to take advantage of the feature. For more information, see [Windows IAM roles for tasks \(p. 846\)](#).

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 \(p. 691\)](#).

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 Amazon EC2 Linux instances, the valid values are `none`, `bridge`, `awsvpc`, and `host`. If no network mode is specified, the default network mode is `bridge`. For Amazon ECS tasks hosted on Amazon EC2 Windows instances, the valid values are `default`, and `awsvpc`. If no network mode is specified, the default network mode is used.

If the network mode is set to `none`, the task's containers do not have external connectivity and port mappings can't be specified in the container definition.

If the network mode is `bridge`, the task utilizes Docker's built-in virtual network which runs inside each container instance.

If the network mode is `host`, the task bypasses Docker's built-in virtual network and maps container ports directly to the Amazon EC2 instance's network interface. In this mode, you can't run multiple instantiations of the same task on a single container instance when port mappings are used.

Important

When using the `host` network mode, you should not run containers using the root user (UID 0). It is considered best practice to use a non-root user.

If the network mode is `awsvpc`, the task is allocated an elastic network interface, and you must specify a `NetworkConfiguration` when you create a service or run a task with the task definition. For more information, see [Amazon ECS task networking \(p. 278\)](#). Currently, only the Amazon ECS-optimized AMI, other Amazon Linux variants with the `ecs-init` package, or AWS Fargate infrastructure support the `awsvpc` network mode.

The `host` and `awsvpc` network modes offer the highest networking performance for containers because they use the Amazon EC2 network stack instead of the virtualized network stack provided by the `bridge` mode. With the `host` and `awsvpc` network modes, exposed container ports are mapped directly to the corresponding host port (for the `host` network mode) or the attached elastic network interface port (for the `awsvpc` network mode), so you cannot take advantage of dynamic host port mappings.

If using the Fargate launch type, the `awsvpc` network mode is required. If using the EC2 launch type, the allowable network mode depends on the underlying EC2 instance's operating system. If Linux, any network mode can be used. If Windows, the `default`, and `awsvpc` modes can be used.

Runtime platform

The following parameter are required for Fargate launch types.

`operatingSystemFamily`

Type: string

Required: Conditional

Default: Linux

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

When you register a task definition, you specify the operating system family. The valid values are `LINUX`, `WINDOWS_SERVER_2019_FULL`, and `WINDOWS_SERVER_2019_CORE`.

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

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

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 hosted on Amazon EC2 instances, these fields are optional. For tasks 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 hosted on Amazon EC2 instances, this field is optional. If your cluster does not have any registered container instances with the requested CPU units available, the task will fail. Supported values are between 128 CPU units (0.125 vCPUs) and 10240 CPU units (10 vCPUs).

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 `memory` parameter:

CPU value	Memory value	Operating systems supported for Fargate
256 (.25 vCPU)	512 MB, 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

CPU value	Memory value	Operating systems supported for Fargate
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 Amazon EC2 instances, this field is optional and any value can be used. If a task-level memory value is specified then the container-level memory value is optional. If your cluster does not have any registered container instances with the requested memory available, the task will fail. If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Container Instance Memory Management \(p. 382\)](#).

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 \(p. 214\)](#)
- [Advanced container definition parameters \(p. 218\)](#)
- [Other container definition parameters \(p. 232\)](#)

Standard container definition parameters

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

Topics

- [Name \(p. 214\)](#)
- [Image \(p. 214\)](#)
- [Memory \(p. 215\)](#)
- [Port mappings \(p. 216\)](#)

Name

`name`

Type: string

Required: yes

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

Image

`image`

Type: string

Required: yes

The image used to start a container. This string is passed directly to the Docker daemon. Images in the Docker Hub registry are available by default. You can also specify other repositories with either `repository-url/image:tag` or `repository-url/image@digest`. Up to 255 letters (uppercase and lowercase), numbers, hyphens, underscores, colons, periods, forward slashes, and number signs are allowed. This parameter maps to `Image` in the [Create a container](#) section of the [Docker Remote API](#) and the `IMAGE` parameter of `docker run`.

- When a new task starts, the Amazon ECS container agent pulls the latest version of the specified image and tag for the container to use. However, subsequent updates to a repository image are not propagated to already running tasks.
- Images in private registries are supported. For more information, see [Private registry authentication for tasks \(p. 300\)](#).
- 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@sha256:94af1f2e64d908bc90dbca0035a5b567EXAMPLE`
- 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 required.

If using the EC2 launch type, you must specify either a task-level memory value or a container-level memory value. If you specify both a container-level `memory` and `memoryReservation` value, `memory` must be greater than `memoryReservation`. If you specify `memoryReservation`, then that value is subtracted from the available memory resources for the container instance on which the container is placed. Otherwise, the value of `memory` is used.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container, so you should not specify fewer than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container, so you should not specify fewer than 4 MiB of memory for your containers.

Note

If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Container Instance Memory Management \(p. 382\)](#).

`memoryReservation`

Type: integer

Required: no

The soft limit (in MiB) of memory to reserve for the container. When system memory is under contention, Docker attempts to keep the container memory to this soft limit; however, your container can consume more memory when needed, up to either the hard limit specified with the `memory` parameter (if applicable), or all of the available memory on the container instance, whichever comes first. This parameter maps to `MemoryReservation` in the [Create a container](#) section of the [Docker Remote API](#) and the `--memory-reservation` option to [docker run](#).

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

For example, if your container normally uses 128 MiB of memory, but occasionally bursts to 256 MiB of memory for short periods of time, you can set a `memoryReservation` of 128 MiB, and a `memory` hard limit of 300 MiB. This configuration would allow the container to only reserve 128 MiB

of memory from the remaining resources on the container instance, but also allow the container to consume more memory resources when needed.

The Docker 20.10.0 or later daemon reserves a minimum of 6 MiB of memory for a container, so you should not specify fewer than 6 MiB of memory for your containers.

The Docker 19.03.13-ce or earlier daemon reserves a minimum of 4 MiB of memory for a container, so you should not specify fewer than 4 MiB of memory for your containers.

Port mappings

`portMappings`

Type: object array

Required: no

Port mappings allow containers to access ports on the host container instance to send or receive traffic.

For task definitions that use the `awsvpc` network mode, you should only specify the `containerPort`. The `hostPort` can be left blank or it must be the same value as the `containerPort`.

Port mappings on Windows use the `NetNAT` gateway address rather than `localhost`. There is no loopback for port mappings on Windows, so you cannot access a container's mapped port from the host itself.

This parameter maps to `PortBindings` in the [Create a container](#) section of the [Docker Remote API](#) and the `--publish` option to [`docker run`](#). If the network mode of a task definition is set to host, then host ports must either be undefined or they must match the container port in the port mapping.

Note

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

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

`containerPort`

Type: integer

Required: yes, when `portMappings` are used

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

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

For Windows containers on Fargate, you cannot use port 3150 for the `containerPort`, because it is 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 do not count toward the 100 reserved ports limit of a container instance.

hostPort

Type: integer

Required: no

The port number on the container instance to reserve for your container.

If using containers in a task with the Fargate launch type, the `hostPort` can either be left blank or be the same value as `containerPort`.

If using containers in a task with the EC2 launch type, you can specify a non-reserved host port for your container port mapping (this is referred to as *static* host port mapping), or you can omit the `hostPort` (or set it to 0) while specifying a `containerPort` and your container automatically receives a port (this is referred to as *dynamic* host port mapping) in the ephemeral port range for your container instance operating system and Docker version.

The default ephemeral port range Docker version 1.6.0 and later is listed on the instance under `/proc/sys/net/ipv4/ip_local_port_range`. If this kernel parameter is unavailable, the default ephemeral port range from 49153–65535 is used. Do not attempt to specify a host port in the ephemeral port range, as these are reserved for automatic assignment. In general, ports below 32768 are outside of the ephemeral port range.

The default reserved ports are 22 for SSH, the Docker ports 2375 and 2376, and the Amazon ECS container agent ports 51678–51680. Any host port that was previously user-specified for a running task is also reserved while the task is running (after a task stops, the host port is released). The current reserved ports are displayed in the `remainingResources` of **describe-container-instances** output, and a container instance may have up to 100 reserved ports at a time, including the default reserved ports. Automatically assigned ports do not count toward the 100 reserved ports limit.

protocol

Type: string

Required: no

The protocol used for the port mapping. Valid values are `tcp` and `udp`. The default is `tcp`.

Important

UDP support is only available on container instances that were launched with version 1.2.0 of the Amazon ECS container agent (such as the `amzn-ami-2015.03.c-amazon-ecs-optimized` AMI) or later, or with container agents that have been updated to version 1.3.0 or later. To update your container agent to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

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

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

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

```
"portMappings": [
  {
```

```
        "containerPort": integer
    }
    ...
]
```

Advanced container definition parameters

The following advanced container definition parameters provide extended capabilities to the [docker run](#) command that is used to launch containers on your Amazon ECS container instances.

Topics

- [Health check \(p. 218\)](#)
- [Environment \(p. 219\)](#)
- [Network settings \(p. 223\)](#)
- [Storage and logging \(p. 225\)](#)
- [Security \(p. 230\)](#)
- [Resource limits \(p. 231\)](#)
- [Docker labels \(p. 232\)](#)

Health check

healthCheck

The container health check command and associated configuration parameters for the container. This parameter maps to `HealthCheck` in the [Create a container](#) section of the [Docker Remote API](#) and the `HEALTHCHECK` parameter of [docker run](#).

Note

The Amazon ECS container agent only monitors and reports on the health checks specified in the task definition. Amazon ECS does not monitor Docker health checks that are embedded in a container image and not specified in the container definition. Health check parameters that are specified in a container definition override any Docker health checks that exist in the container image.

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

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

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

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

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

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

The following are notes about container health check support:

- Container health checks require version 1.17.0 or greater of the Amazon ECS container agent. For more information, see [Updating the Amazon ECS container agent \(p. 448\)](#).
- Container health checks are supported for Fargate tasks if you are using platform version 1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 169\)](#).
- Container health checks are not supported for tasks that are part of a service that is configured to use a Classic Load Balancer.

command

A string array representing the command that the container runs to determine if it is healthy.

The string array can start with `CMD` to execute the command arguments directly, or `CMD-SHELL` to run the command with the container's default shell. If neither is specified, `CMD` is used by default.

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

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

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

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

An exit code of 0 indicates success, and a non-zero exit code indicates failure. For more information, see `HealthCheck` in the [Create a container](#) section of the [Docker Remote API](#).

interval

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

timeout

The time period in seconds to wait for a health check to succeed before it is considered a failure. You may specify between 2 and 60 seconds. The default value is 5 seconds.

retries

The number of times to retry a failed health check before the container is considered unhealthy. You may specify between 1 and 10 retries. The default value is three retries.

startPeriod

The optional grace period within which to provide containers time to bootstrap before failed health checks count towards the maximum number of retries. You may specify between 0 and 300 seconds. The `startPeriod` is disabled by default.

Environment

cpu

Type: integer

Required: conditional

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

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

Note

You can determine the number of CPU units that are available per Amazon EC2 instance type by multiplying the number of vCPUs listed for that instance type on the [Amazon EC2 Instances](#) detail page by 1,024.

Linux containers share unallocated CPU units with other containers on the container instance with the same ratio as their allocated amount. For example, if you run a single-container task on a single-core instance type with 512 CPU units specified for that container, and that is the only task running on the container instance, that container could use the full 1,024 CPU unit share at any given time. However, if you launched another copy of the same task on that container instance, each task would be guaranteed a minimum of 512 CPU units when needed, and each container could float to higher CPU usage if the other container was not using it, but if both tasks were 100% active all of the time, they would be limited to 512 CPU units.

On Linux container instances, the Docker daemon on the container instance uses the `CPU` value to calculate the relative CPU share ratios for running containers. For more information, see [CPU share constraint](#) in the Docker documentation. The minimum valid CPU share value that the Linux kernel allows is 2. However, the `CPU` parameter is not required, and you can use CPU values below 2 in your container definitions. For CPU values below 2 (including null), the behavior varies based on your Amazon ECS container agent version:

- **Agent versions <= 1.1.0:** Null and zero CPU values are passed to Docker as 0, which Docker then converts to 1,024 CPU shares. CPU values of 1 are passed to Docker as 1, which the Linux kernel converts to two CPU shares.
- **Agent versions >= 1.2.0:** Null, zero, and CPU values of 1 are passed to Docker as two CPU shares.

On Windows container instances, the `CPU` limit is enforced as an absolute limit, or a quota. Windows containers only have access to the specified amount of CPU that is 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](#).

`gpu`

Type: [ResourceRequirement](#) object

Required: no

The number of physical GPUs the Amazon ECS container agent will reserve for the container. The number of GPUs reserved for all containers in a task should not exceed the number of available GPUs on the container instance the task is launched on. For more information, see [Working with GPUs on Amazon ECS \(p. 250\)](#).

Note

This parameter is not supported for Windows containers or containers hosted on Fargate.

`essential`

Type: Boolean

Required: no

If the `essential` parameter of a container is marked as `true`, and that container fails or stops for any reason, all other containers that are part of the task are stopped. If the `essential` parameter

of a container is marked as `false`, then its failure does not affect the rest of the containers in a task. If this parameter is omitted, a container is assumed to be essential.

All tasks must have at least one essential container. If you have an application that is composed of multiple containers, you should group containers that are used for a common purpose into components, and separate the different components into multiple task definitions. For more information, see [Application architecture \(p. 195\)](#).

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

entryPoint

Important

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

Type: string array

Required: no

The entry point that is passed to the container. This parameter maps to `Entrypoint` in the [Create a container](#) section of the [Docker Remote API](#) and the `--entrypoint` option to [docker run](#). For more information about the Docker `ENTRYPOINT` parameter, go to <https://docs.docker.com/engine/reference/builder/#entrypoint>.

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

command

Type: string array

Required: no

The command that is passed to the container. This parameter maps to `Cmd` in the [Create a container](#) section of the [Docker Remote API](#) and the `COMMAND` parameter to [docker run](#). For more information about the Docker `CMD` parameter, go to <https://docs.docker.com/engine/reference/builder/#cmd>. If there are multiple arguments, each argument should be a separated string in the array.

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

workingDirectory

Type: string

Required: no

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

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

environmentFiles

Type: object array

Required: no

A list of files containing the environment variables to pass to a container. This parameter maps to the `--env-file` option to [docker run](#).

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

If there are individual environment variables specified in the container definition, they take precedence over the variables contained within an environment file. If multiple environment files are specified that contain the same variable, they are processed from the top down. It is recommended to use unique variable names. For more information, see [Specifying environment variables \(p. 315\)](#).

`value`

Type: String

Required: Yes

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

`type`

Type: String

Required: Yes

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

`environment`

Type: object array

Required: no

The environment variables to pass to a container. This parameter maps to `Env` in the [Create a container](#) section of the [Docker Remote API](#) and the `--env` option to [docker run](#).

Important

We do not recommend using plaintext environment variables for sensitive information, such as credential data.

`name`

Type: String

Required: Yes, when `environment` is used

The name of the environment variable.

`value`

Type: String

Required: Yes, when `environment` is used

The value of the environment variable.

```
"environment" : [
```

```
        { "name" : "string", "value" : "string" },
        { "name" : "string", "value" : "string" }
    ]
```

secrets

Type: Object array

Required: No

An object representing the secret to expose to your container. For more information, see [Specifying sensitive data \(p. 303\)](#).

name

Type: String

Required: Yes

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

valueFrom

Type: String

Required: Yes

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

Note

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

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

Network settings

disableNetworking

Type: Boolean

Required: no

When this parameter is true, networking is off within the container. This parameter maps to NetworkDisabled in the [Create a container](#) section of the [Docker Remote API](#).

Note

This parameter is not supported for Windows containers or tasks using the awsvpc network mode.

```
"disableNetworking": true|false
```

links

Type: string array

Required: no

The `link` parameter allows containers to communicate with each other without the need for port mappings. Only supported if the network mode of a task definition is set to `bridge`. The `name:internalName` construct is analogous to `name:alias` in Docker links. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. For more information about linking Docker containers, go to https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/. This parameter maps to `Links` in the [Create a container](#) section of the [Docker Remote API](#) and the `--link` option to `docker run`.

Note

This parameter is not supported for Windows containers or tasks using the `awsvpc` network mode.

Important

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

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

hostname

Type: string

Required: no

The hostname to use for your container. This parameter maps to `Hostname` in the [Create a container](#) section of the [Docker Remote API](#) and the `--hostname` option to `docker run`.

Note

The `hostname` parameter is not supported if you are using the `awsvpc` network mode.

```
"hostname": "string"
```

dnsServers

Type: string array

Required: no

A list of DNS servers that are presented to the container. This parameter maps to `Dns` in the [Create a container](#) section of the [Docker Remote API](#) and the `--dns` option to `docker run`.

Note

This parameter is not supported for Windows containers or tasks using the `awsvpc` network mode.

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

dnsSearchDomains

Type: string array

Required: no

Pattern: ^[a-zA-Z0-9-]{0,253}[a-zA-Z0-9]\$

A list of DNS search domains that are presented to the container. This parameter maps to DnsSearch in the [Create a container](#) section of the [Docker Remote API](#) and the --dns-search option to [docker run](#).

Note

This parameter is not supported for Windows containers or tasks using the awsvpc network mode.

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

extraHosts

Type: object array

Required: no

A list of hostnames and IP address mappings to append to the /etc/hosts file on the container.

This parameter maps to ExtraHosts in the [Create a container](#) section of the [Docker Remote API](#) and the --add-host option to [docker run](#).

Note

This parameter is not supported for Windows containers or tasks that use the awsvpc network mode.

```
"extraHosts": [
    {
        "hostname": "string",
        "ipAddress": "string"
    }
    ...
]
```

hostname

Type: string

Required: yes, when extraHosts are used

The hostname to use in the /etc/hosts entry.

ipAddress

Type: string

Required: yes, when extraHosts are used

The IP address to use in the /etc/hosts entry.

Storage and logging

readonlyRootFilesystem

Type: Boolean

Required: no

When this parameter is true, the container is given read-only access to its root file system. This parameter maps to ReadonlyRootfs in the [Create a container](#) section of the [Docker Remote API](#) and the --read-only option to [docker run](#).

Note

This parameter is not supported for Windows containers.

```
"readonlyRootFilesystem": true|false
```

mountPoints

Type: Object Array

Required: No

The mount points for data volumes in your container.

This parameter maps to `Volumes` in the [Create a container](#) section of the [Docker Remote API](#) and the `--volume` option to `docker run`.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives.

sourceVolume

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

containerPath

Type: String

Required: Yes, when `mountPoints` are used

The path on the container to mount the volume at.

readOnly

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

volumesFrom

Type: Object Array

Required: No

Data volumes to mount from another container. This parameter maps to `VolumesFrom` in the [Create a container](#) section of the [Docker Remote API](#) and the `--volumes-from` option to `docker run`.

sourceContainer

Type: string

Required: yes, when `volumesFrom` is used

The name of the container to mount volumes from.

readOnly

Type: Boolean

Required: no

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

```
"volumesFrom": [
    {
        "sourceContainer": "string",
        "readOnly": true|false
    }
]
```

logConfiguration

Type: [LogConfiguration Object](#)

Required: no

The log configuration specification for the container.

For example task definitions using a log configuration, see [Example task definitions \(p. 317\)](#).

This parameter maps to `LogConfig` in the [Create a container](#) section of the [Docker Remote API](#) and the `--log-driver` option to `docker run`. By default, containers use the same logging driver that the Docker daemon uses; however the container may use a different logging driver than the Docker daemon by specifying a log driver with this parameter in the container definition. To use a different logging driver for a container, the log system must be configured properly on the container instance (or on a different log server for remote logging options). For more information on the options for different supported log drivers, see [Configure logging drivers](#) in the Docker documentation.

The following should be noted when specifying a log configuration for your containers:

- Amazon ECS currently supports a subset of the logging drivers available to the Docker daemon (shown in the valid values below). Additional log drivers may be available in future releases of the Amazon ECS container agent.
- This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.
- For tasks using the EC2 launch type, the Amazon ECS container agent running on a container instance must register the logging drivers available on that instance with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable before containers placed on that instance can use these log configuration options. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).
- For tasks using the Fargate launch type, because you do not have access to the underlying infrastructure your tasks are hosted on, any additional software needed will have to be installed outside of the task. For example, the Fluentd output aggregators or a remote host running Logstash to send Gelf logs to.

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

}

logDriver

Type: string

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

Required: yes, when logConfiguration is used

The log driver to use for the container. The valid values listed earlier are log drivers that the Amazon ECS container agent can communicate with by default.

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

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

For more information on using the awslogs log driver in task definitions to send your container logs to CloudWatch Logs, see [Using the awslogs log driver \(p. 283\)](#).

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

Note

If you have a custom driver that is not listed, you can fork the Amazon ECS container agent project that is [available on GitHub](#) and customize it to work with that driver.

We encourage you to submit pull requests for changes that you would like to have included. However, we do not currently provide support for running modified copies of this software.

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

options

Type: string to string map

Required: no

The configuration options to send to the log driver.

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

secretOptions

Type: object array

Required: no

An object representing the secret to pass to the log configuration. For more information, see [Specifying sensitive data \(p. 303\)](#).

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 \(p. 289\)](#).

```
{  
    "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 \(p. 292\)](#).

type

Type: String

Required: Yes

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

Security

`privileged`

Type: Boolean

Required: no

When this parameter is true, the container is given elevated privileges on the host container instance (similar to the root user).

This parameter maps to `Privileged` in the [Create a container](#) section of the [Docker Remote API](#) and the `--privileged` option to [docker run](#).

Note

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

```
"privileged": true|false
```

`user`

Type: string

Required: no

The user 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](#).

Important

When running tasks using the `host` network mode, you should not run containers using the root user (UID 0). It is considered best practice to use a non-root user.

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

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

Note

This parameter is not supported for Windows containers.

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

`dockerSecurityOptions`

Type: string array

Valid values: `"no-new-privileges"` | `"apparmor:PROFILE"` | `"label:value"` | `"credentialspec:credentialSpecFilePath"`

Required: no

A list of strings to provide custom labels for SELinux and AppArmor multi-level security systems. For more information about valid values, see [Docker Run Security Configuration](#). This field is not valid for containers in tasks using the Fargate launch type.

With Windows containers, this parameter can be used to reference a credential spec file when configuring a container for Active Directory authentication. For more information, see [Using gMSAs for Windows Containers \(p. 849\)](#).

This parameter maps to `SecurityOpt` in the [Create a container](#) section of the [Docker Remote API](#) and the `--security-opt` option to [docker](#).

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

Note

The Amazon ECS container agent running on a container instance must register with the `ECS_SELINUX_CAPABLE=true` or `ECS_APPARMOR_CAPABLE=true` environment variables before containers placed on that instance can use these security options. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

Resource limits

ulimits

Type: object array

Required: no

A list of `ulimit` values to define for a container. This value would overwrite the default resource limit 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. For more information, see [Task resource limits \(p. 163\)](#).

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" | "fsiz" | "locks" | "memlock" | "msgqueue" | "nice" | "nofile" | "nproc" | "rss" | "rtprio" | "rttime" | "sigpending" | "stack"
        "softLimit": integer,
        "hardLimit": integer
    }
    ...
]
```

name

Type: string

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

Required: yes, when `ulimits` are used

The type of the ulimit.

`hardLimit`

Type: integer

Required: yes, when ulimits are used

The hard limit for the ulimit type.

`softLimit`

Type: integer

Required: yes, when ulimits are used

The soft limit for the ulimit type.

Docker labels

`dockerLabels`

Type: string to string map

Required: no

A key/value map of labels to add to the container. This parameter maps to `Labels` in the [Create a container](#) section of the [Docker Remote API](#) and the `--label` option to [docker run](#).

This parameter requires version 1.18 of the Docker Remote API or greater on your container instance.

```
"dockerLabels": {"string": "string"
    ...}
```

Other container definition parameters

The following container definition parameters are able to be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

Topics

- [Linux parameters \(p. 232\)](#)
- [Container dependency \(p. 236\)](#)
- [Container timeouts \(p. 237\)](#)
- [System controls \(p. 238\)](#)
- [Interactive \(p. 239\)](#)
- [Pseudo terminal \(p. 239\)](#)

Linux parameters

`linuxParameters`

Type: [LinuxParameters](#) object

Required: no

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

Note

This parameter is not supported for Windows containers.

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

capabilities

Type: [KernelCapabilities](#) object

Required: no

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

add

Type: string array

Valid values: "ALL" | "AUDIT_CONTROL" | "AUDIT_READ" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE" | "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" | "SYSLOG" | "WAKE_ALARM"

Required: no

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

Note

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

drop

Type: string array

Valid values: "ALL" | "AUDIT_CONTROL" | "AUDIT_WRITE" | "BLOCK_SUSPEND" | "CHOWN" | "DAC_OVERRIDE" | "DAC_READ_SEARCH" | "FOWNER" | "FSETID" | "IPC_LOCK" | "IPC_OWNER" | "KILL" | "LEASE" | "LINUX_IMMUTABLE" | "MAC_ADMIN" | "MAC_OVERRIDE" | "MKNOD" | "NET_ADMIN" | "NET_BIND_SERVICE" | "NET_BROADCAST" | "NET_RAW" | "SETFCAP" | "SETGID" | "SETPCAP" | "SETUID" | "SYS_ADMIN" | "SYS_BOOT" | "SYS_CHROOT" | "SYS_MODULE" | "SYS_NICE" | "SYS_PACCT" | "SYS_PTRACE"

| "SYS_RAWIO" | "SYS_RESOURCE" | "SYS_TIME" | "SYS_TTY_CONFIG" |
"SYSLOG" | "WAKE_ALARM"

Required: no

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

`devices`

Any host devices to expose to the container. This parameter maps to `Devices` in the [Create a container](#) section of the [Docker Remote API](#) and the `--device` option to `docker run`.

Note

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

Type: Array of [Device](#) objects

Required: No

`hostPath`

The path for the device on the host container instance.

Type: String

Required: Yes

`containerPath`

The path inside the container at which to expose the host device.

Type: String

Required: No

`permissions`

The explicit permissions to provide to the container for the device. By default, the container has permissions for `read`, `write`, and `mknod` on the device.

Type: Array of strings

Valid Values: `read` | `write` | `mknod`

`initProcessEnabled`

Run an `init` process inside the container that forwards signals and reaps processes. This parameter maps to the `--init` option to `docker run`.

This parameter requires version 1.25 of the Docker Remote API or greater on your container instance.

`maxSwap`

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

If a `maxSwap` value of 0 is specified, the container will not use swap. Accepted values are 0 or any positive integer. If the `maxSwap` parameter is omitted, the container will use the swap

configuration for the container instance it is running on. A `maxSwap` value must be set for the `swappiness` parameter to be used.

Note

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

`sharedMemorySize`

The value for the size (in MiB) of the `/dev/shm` volume. This parameter maps to the `--shm-size` option to [docker run](#).

Note

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

Type: Integer

`swappiness`

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

Note

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

`tmpfs`

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

Note

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

Type: Array of [Tmpfs](#) objects

Required: No

`containerPath`

The absolute file path where the `tmpfs` volume is to be mounted.

Type: String

Required: Yes

`mountOptions`

The list of `tmpfs` volume mount options.

Type: Array of strings

Required: No

Valid Values: "defaults" | "ro" | "rw" | "suid" | "nosuid" | "dev" | "nodev" | "exec" | "noexec" | "sync" | "async" | "dirsync" | "remount" | "mand" | "nomand" | "atime" | "noatime" | "diratime" | "nodiratime" | "bind" | "rbind" | "unbindable" | "runbindable" |

```
"private" | "rprivate" | "shared" | "rshared" | "slave" | "rslave" |  
"relatime" | "norelatime" | "strictatime" | "nostrictatime" | "mode"  
| "uid" | "gid" | "nr_inodes" | "nr_blocks" | "mpol"  
size
```

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

Type: Integer

Required: Yes

Container dependency

dependsOn

Type: Array of [ContainerDependency](#) objects

Required: no

The dependencies defined for container startup and shutdown. A container can contain multiple dependencies. When a dependency is defined for container startup, for container shutdown it is reversed. For an example, see [Example: Container dependency \(p. 321\)](#).

Note

If a container does not 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 hosted on Amazon EC2 instances, the instances require at least version 1.26.0 of the container agent to enable container dependencies. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#). If you are using an Amazon ECS-optimized Amazon Linux AMI, your instance needs at least version 1.26.0-1 of the `ecs-init` package. If your container instances are launched from version 20190301 or later, then they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).

For Amazon ECS tasks 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 nonessential containers that run a script and then exit. This condition cannot be set on an essential container.
- **SUCCESS** – This condition is the same as **COMPLETE**, but it also requires that the container exits with a zero status. This condition cannot be set on an essential container.
- **HEALTHY** – This condition validates that the dependent container passes its Docker healthcheck before permitting other containers to start. This requires that the dependent container has health checks configured. This condition is confirmed only at task startup.

Container timeouts

`startTimeout`

Type: Integer

Required: no

Example values: 120

Time duration (in seconds) to wait before giving up on resolving dependencies for a container.

For example, you specify two containers in a task definition with `containerA` having a dependency on `containerB` reaching a **COMPLETE**, **SUCCESS**, or **HEALTHY** status. If a `startTimeout` value is specified for `containerB` and it doesn't reach the desired status within that time then `containerA` will give up and not start.

Note

If a container does not 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 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 using 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 and if the parameter is not specified, the default value of 30 seconds is used.

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

latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#). If you are using an Amazon ECS-optimized Amazon Linux AMI, your instance needs at least version 1.26.0-1 of the `ecs-init` package. If your container instances are launched from version 20190301 or later, then they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).

System controls

`systemControls`

Type: [SystemControl](#) object

Required: no

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

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

- For tasks that use the `awsvpc` network mode, if you set `systemControls` for any container it will apply to all containers in the task. If you set different `systemControls` for multiple containers in a single task, the container that is started last will determine which `systemControls` take effect.
- For tasks that use the host network mode, the network namespace `systemControls` are not supported.

If you are setting an IPC resource namespace to use for the containers in the task, the following will apply to your system controls. For more information, see [IPC mode \(p. 246\)](#).

- For tasks that use the host IPC mode, IPC namespace `systemControls` are not supported.
- For tasks that use the task IPC mode, IPC namespace `systemControls` values will apply to all containers within a task.

Note

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

```
"systemControls": [  
    {  
        "namespace": "string",  
        "value": "string"  
    }  
]
```

`namespace`

Type: String

Required: no

The namespaced kernel parameter to set a value for.

Valid IPC namespace values: `"kernel.msgmax"` | `"kernel.msgmnb"` | `"kernel.msgmni"` | `"kernel.sem"` | `"kernel.shmall"` | `"kernel.shmmmax"` | `"kernel.shmmnmi"` | `"kernel.shm_rmid_forced"`, as well as Sysctls beginning with `"fs.mqueue.*"`

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

`value`

Type: String

Required: no

The value for the namespaced kernel parameter specified in `namespace`.

Interactive

`interactive`

Type: Boolean

Required: no

When this parameter is `true`, this allows you to deploy containerized applications that require `stdin` or a `tty` to be allocated. This parameter maps to `OpenStdin` in the [Create a container](#) section of the [Docker Remote API](#) and the `--interactive` option to [docker run](#).

Pseudo terminal

`pseudoTerminal`

Type: Boolean

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](#).

Task placement constraints

When you register a task definition, you can provide task placement constraints that customize how Amazon ECS places tasks.

If you are using the Fargate launch type, task placement constraints are not supported. By default Fargate tasks are spread across Availability Zones.

For tasks that use the EC2 launch type, you can use constraints to place tasks based on Availability Zone, instance type, or custom attributes. For more information, see [Amazon ECS task placement constraints \(p. 516\)](#).

The following parameters are allowed in a container definition:

`expression`

Type: string

Required: no

A cluster query language expression to apply to the constraint. For more information, see [Cluster query language \(p. 520\)](#).

`type`

Type: string

Required: yes

The type of constraint. Use `memberOf` to restrict the selection to a group of valid candidates.

Proxy configuration

`proxyConfiguration`

Type: [ProxyConfiguration](#) object

Required: no

The configuration details for the App Mesh proxy.

For tasks using the EC2 launch type, the container instances require at least version 1.26.0 of the container agent and at least version 1.26.0-1 of the `ecs-init` package to enable a proxy configuration. If your container instances are launched from the Amazon ECS-optimized AMI version 20190301 or later, then they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).

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

Note

This parameter is not supported for Windows containers.

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

`type`

Type: String

Value values: APPMESH

Required: No

The proxy type. The only supported value is APPMESH.

`containerName`

Type: String

Required: Yes

The name of the container that will serve as the App Mesh proxy.

`properties`

Type: Array of [KeyValuePair](#) objects

Required: No

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

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

`name`

Type: String

Required: No

The name of the key-value pair.

`value`

Type: String

Required: No

The value of the key-value pair.

Volumes

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

The following are the types of data volumes that can be used:

- Docker volumes — A Docker-managed volume that is created under `/var/lib/docker/volumes` on the host Amazon EC2 instance. Docker volume drivers (also referred to as plugins) are used to integrate the volumes with external storage systems, such as Amazon EBS. The built-in local volume driver or a third-party volume driver can be used. Docker volumes are only supported when running tasks on Amazon EC2 instances. Windows containers only support the use of the local driver. To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see [Using volumes](#).
- Bind mounts — A file or directory on the host machine is mounted into a container. Bind mount host volumes are supported when running tasks on either AWS Fargate or Amazon EC2 instances. To use bind mount host volumes, specify a `host` and optional `sourcePath` value in your task definition. For more information, see [Using bind mounts](#).

For more information, see [Using data volumes in tasks \(p. 256\)](#).

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.

host

Required: No

Note

The `host` parameter is only supported when using tasks hosted on Amazon EC2 instances.

The `host` parameter is used to tie the lifecycle of the bind mount to the host Amazon EC2 instance, rather than the task, and where it is stored. If the `host` parameter is empty, then the Docker daemon assigns a host path for your data volume, but the data is not guaranteed to persist after the containers associated with it stop running.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`.

sourcePath

Type: String

Required: No

When the `host` parameter is used, specify a `sourcePath` to declare the path on the host Amazon EC2 instance that is presented to the container. If this parameter is empty, then the Docker daemon assigns a host path for you. If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host Amazon EC2 instance until you delete it manually. If the `sourcePath` value does not exist on the host Amazon EC2 instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

dockerVolumeConfiguration

Type: Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are only supported when running tasks on EC2 instances. Windows containers only support the use of the `local` driver. To use bind mounts, specify a `host` instead.

scope

Type: String

Valid Values: `task` | `shared`

Required: No

The scope for the Docker volume, which determines its lifecycle. Docker volumes that are scoped to a task are automatically provisioned when the task starts destroyed when the task is cleaned up. Docker volumes that are scoped as `shared` persist after the task stops.

autoprovision

Type: Boolean

Default value: `false`

Required: No

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

`driver`

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because it is used for task placement. If the driver was installed using the Docker plugin CLI, use `docker plugin ls` to retrieve the driver name from your container instance. If the driver was installed using another method, use Docker plugin discovery to retrieve the driver name. For more information, see [Docker plugin discovery](#). This parameter maps to `Driver` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--driver` option to `docker volume create`.

`driverOpts`

Type: String

Required: No

A map of Docker driver specific options to pass through. This parameter maps to `DriverOpts` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--opt` option to `docker volume create`.

`labels`

Type: String

Required: No

Custom metadata to add to your Docker volume. This parameter maps to `Labels` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--label` option to `docker volume create`.

`efsVolumeConfiguration`

Type: Object

Required: No

This parameter is specified when using Amazon EFS volumes.

`fileSystemId`

Type: String

Required: Yes

The Amazon EFS file system ID to use.

`rootDirectory`

Type: String

Required: No

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

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 `rootDirectory` 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](#).

FSxWindowsFileServerVolumeConfiguration

Type: Object

Required: Yes

This parameter is specified when you are using the [FSx for Windows File Server](#) file system for task storage.

fileSystemId

Type: String

Required: Yes

The FSx for Windows File Server file system ID to use.

rootDirectory

Type: String

Required: Yes

The directory within the FSx for Windows File Server file system to mount as the root directory inside the host.

authorizationConfig

credentialsParameter

Type: String

Required: Yes

The authorization credential options.

options:

- Amazon Resource Name (ARN) of an [AWS Secrets Manager](#) secret.
- ARN of an [AWS Systems Manager](#) parameter.

domain

Type: String

Required: Yes

A fully qualified domain name hosted by an [AWS Directory Service](#) Managed Microsoft AD (Active Directory) or self-hosted EC2 AD.

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, both of which you define. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

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.

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 are able to be used when registering task definitions in the Amazon ECS console by using the **Configure via JSON** option. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

Topics

- [Ephemeral storage \(p. 246\)](#)
- [IPC mode \(p. 246\)](#)
- [PID mode \(p. 247\)](#)

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 hosted on AWS Fargate. For more information, see [Fargate task storage \(p. 256\)](#).

Note

This parameter is only supported for tasks hosted on AWS Fargate using platform version 1.4.0 or later (Linux). This is not 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 containers within the tasks that specified the host IPC mode on the same container instance share the same IPC resources with the host Amazon EC2 instance. If task is specified, all containers within the specified task share the same IPC resources. If none is

specified, then IPC resources within the containers of a task are private and not shared with other containers in a task or on the container instance. If no value is specified, then the IPC resource namespace sharing depends on the Docker daemon setting on the container instance. For more information, see [IPC settings](#) in the *Docker run reference*.

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

If you are setting namespaced kernel parameters using `systemControls` for the containers in the task, the following will apply to your IPC resource namespace. For more information, see [System controls \(p. 238\)](#).

- For tasks that use the host IPC mode, IPC namespace related `systemControls` are not supported.
- For tasks that use the task IPC mode, IPC namespace related `systemControls` will apply to all containers within a task.

Note

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

PID mode

`pidMode`

Type: String

Required: No

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

If the host PID mode is used, be aware that there is a heightened risk of undesired process namespace exposure. For more information, see [Docker security](#).

Note

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

Amazon ECS launch types

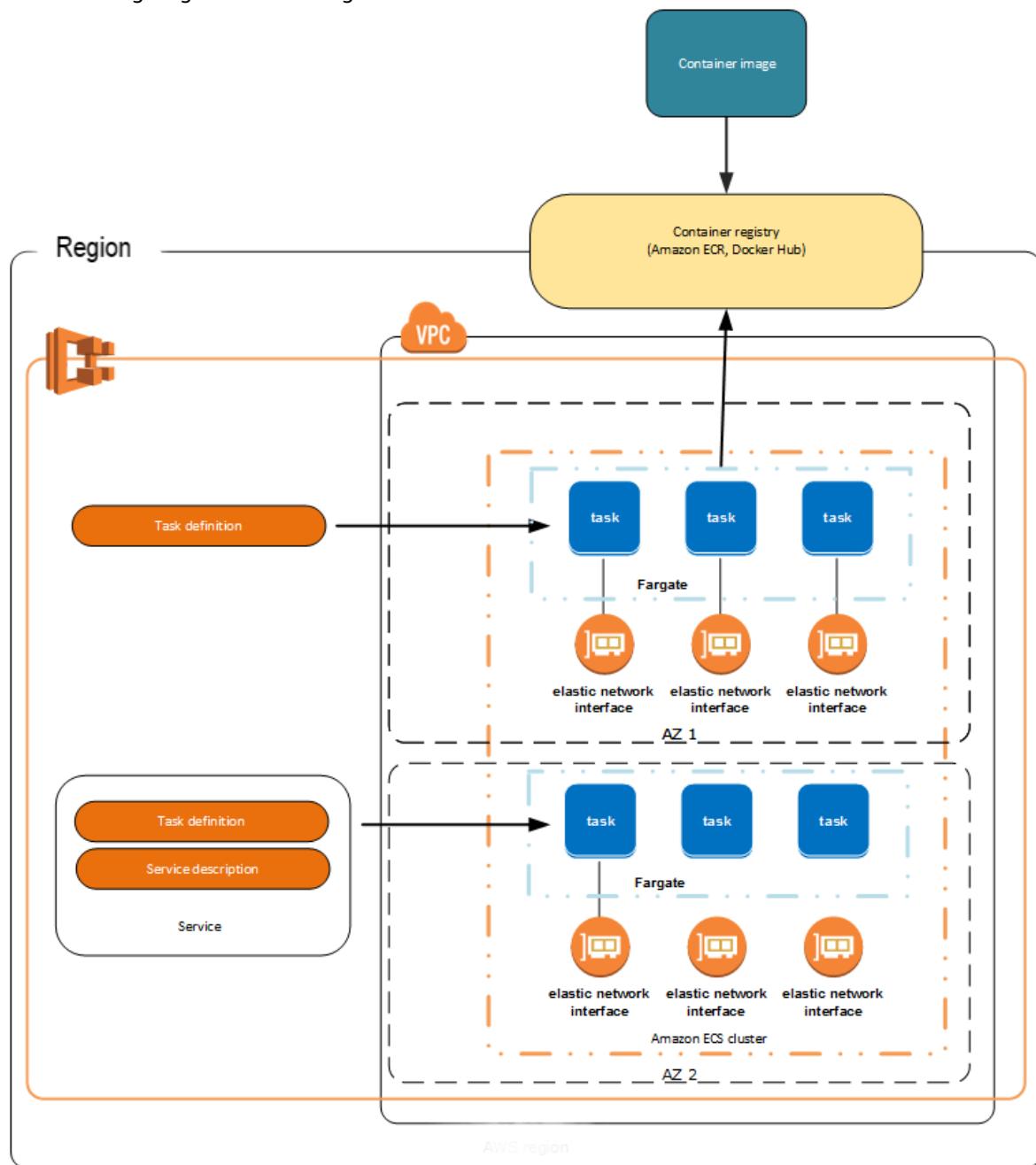
An Amazon ECS launch type can be specified when running a standalone task or creating a service to determine the infrastructure on which your tasks and services are hosted. The following are the available launch types.

Fargate launch type

The Fargate launch type can be used to run your containerized applications without the need to provision and manage the backend 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” \(p. 614\)](#).

The following diagram shows the general architecture:

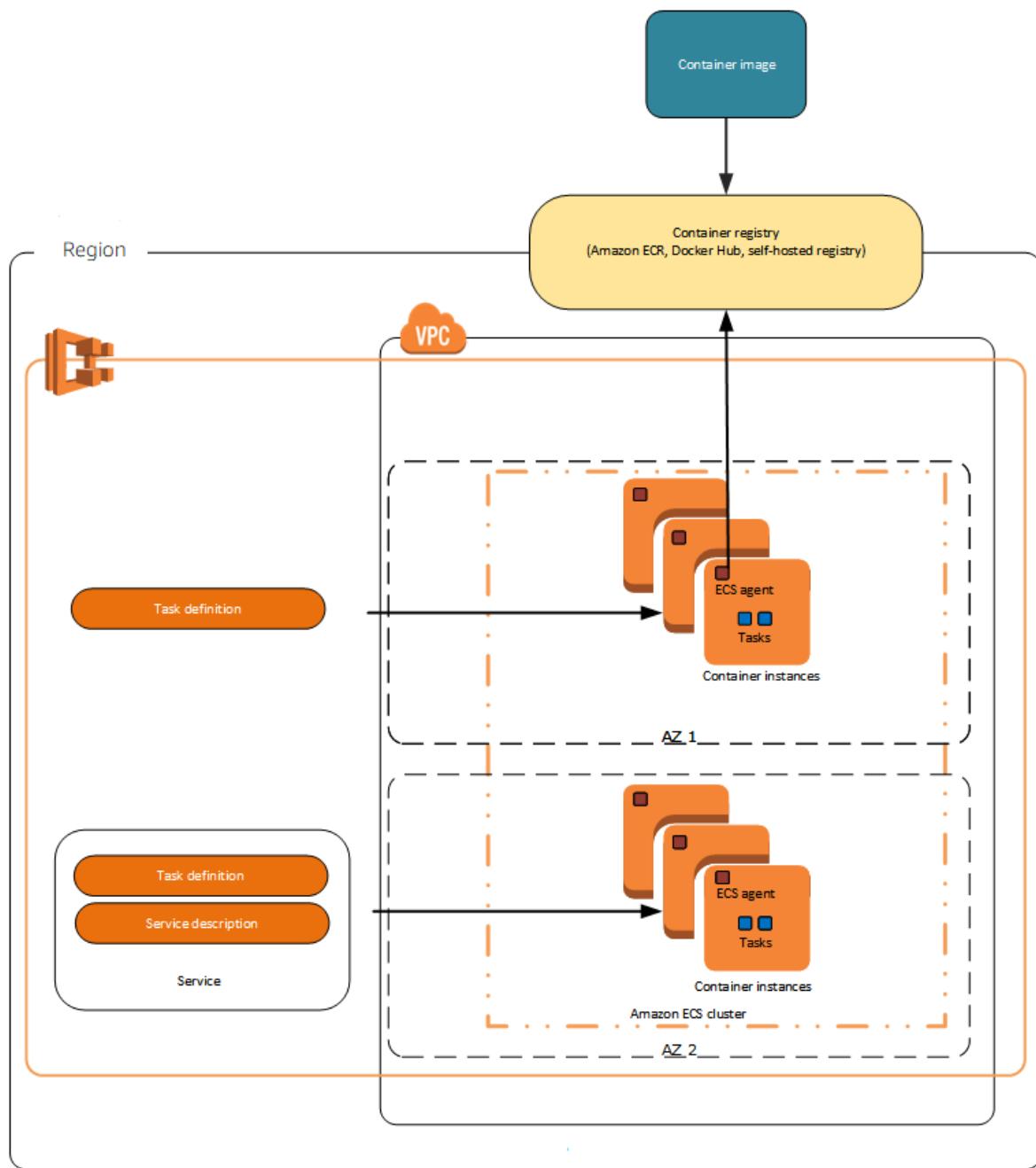


For more information about Amazon ECS on Fargate, see [Amazon ECS on AWS Fargate \(p. 161\)](#).

EC2 launch type

The EC2 launch type can be used to run your containerized applications on Amazon EC2 instances that you register to your Amazon ECS cluster and manage yourself.

The following diagram shows the general architecture:



External launch type

The External launch type is used to run your containerized applications on your on-premise server or virtual machine (VM) that you register to your Amazon ECS cluster and manage remotely. For more information, see [External instances \(Amazon ECS Anywhere\)](#) (p. 414).

Working with GPUs on Amazon ECS

Amazon ECS supports workloads that take advantage of GPUs by enabling you to create clusters with GPU-enabled container instances. Amazon EC2 GPU-based container instances using the p2, p3, g3, and g4 instance types provide access to NVIDIA GPUs. For more information, see [Linux Accelerated Computing Instances in the Amazon EC2 User Guide for Linux Instances](#).

Amazon ECS provides a GPU-optimized AMI that comes ready with pre-configured NVIDIA kernel drivers and a Docker GPU runtime. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).

You can designate a number of GPUs in your task definition for task placement consideration at a container level. Amazon ECS schedules to available GPU-enabled container instances and pin physical GPUs to proper containers for optimal performance.

The following Amazon EC2 GPU-based instance types are supported. For more information, see [Amazon EC2 P2 Instances](#), [Amazon EC2 P3 Instances](#), [Amazon EC2 G3 Instances](#), and [Amazon EC2 G4 Instances](#).

Important

The g4 instance type family is supported on version 20190913 and later of the Amazon ECS GPU-optimized AMI. For more information, see [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#). It is currently not supported in the Create Cluster workflow in the Amazon ECS console. To use these instance types, you must either use the Amazon EC2 console, AWS CLI, or API and manually register the instances to your cluster.

Instance type	GPUs	GPU memory (GiB)	vCPUs	Memory (GiB)
p2.xlarge	1	12	4	61
p2.8xlarge	8	96	32	488
p2.16xlarge	16	192	64	732
p3.2xlarge	1	16	8	61
p3.8xlarge	4	64	32	244
p3.16xlarge	8	128	64	488
p3dn.24xlarge	8	256	96	768
p4d.24xlarge	8	320	96	1152
g3s.xlarge	1	8	4	30.5
g3.4xlarge	1	8	16	122
g3.8xlarge	2	16	32	244
g3.16xlarge	4	32	64	488
g4dn.xlarge	1	16	4	16
g4dn.2xlarge	1	16	8	32
g4dn.4xlarge	1	16	16	64
g4dn.8xlarge	1	16	32	128
g4dn.12xlarge	4	64	48	192

Instance type	GPUs	GPU memory (GiB)	vCPUs	Memory (GiB)
g4dn.16xlarge	1	16	64	256

Topics

- [Considerations \(p. 251\)](#)
- [Specifying GPUs in your task definition \(p. 252\)](#)

Considerations

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

- Your clusters can contain a mix of GPU and non-GPU container instances.
- Running GPU workloads is supported on external instances. When registering an external instance with your cluster, ensure the `--enable-gpu` flag is included on the installation script. For more information, see [Registering an external instance to a cluster \(p. 418\)](#).
- When running a task or creating a service, you can use instance type attributes when configuring task placement constraints to ensure which of your container instances the task is launched on. By doing this, you can more effectively use your resources. For more information, see [Amazon ECS task placement \(p. 513\)](#).

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

```
aws ecs run-task --cluster default --task-definition ecs-gpu-task-def \
    --placement-constraints type=memberOf,expression="attribute:ecs.instance-type == \
    p2.xlarge" --region us-east-2
```

- For each container that has a GPU resource requirement specified in the container definition, Amazon ECS sets the container runtime to be the NVIDIA container runtime.
- The NVIDIA container runtime requires some environment variables to be set in the container in order to work. For a list of these environment variables, see [nvidia-container-runtime](#). Amazon ECS sets the `NVIDIA_VISIBLE_DEVICES` environment variable value to be a list of the GPU device IDs that Amazon ECS assigns to the container. For the other required environment variables, Amazon ECS doesn't set them, so you should ensure that your container image sets them or they should be set in the container definition.
- The g4 instance type family is supported on version 20190913 and later of the Amazon ECS GPU-optimized AMI. For more information, see [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#). It is currently not supported in the Create Cluster workflow in the Amazon ECS console. To use these instance types, you must either use the Amazon EC2 console, AWS CLI, or API and manually register the instances to your cluster.
- The p4d.24xlarge instance type only works with CUDA 11 or later.
- The Amazon ECS GPU-optimized AMI has IPv6 enabled, which causes issues when using `yum`. This can be resolved by configuring `yum` to use IPv4 with the following command.

```
echo "ip_resolve=4" >> /etc/yum.conf
```

- When you build a container image that does not use the NVIDIA/CUDA base images, you must set the `NVIDIA_DRIVER_CAPABILITIES` container runtime variable to one of the following values:
 - `utility,compute`
 - `all`

For information about how to set the variable, see [Controlling the NVIDIA Container Runtime](#) on the NVIDIA website.

Specifying GPUs in your task definition

To take advantage of the GPUs on a container instance and the Docker GPU runtime, ensure you designate the number of GPUs your container requires in the task definition. As GPU-enabled containers are placed, the Amazon ECS container agent pins the desired number of physical GPUs to the appropriate container. The number of GPUs reserved for all containers in a task should not exceed the number of available GPUs on the container instance the task is launched on. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

Important

If your GPU requirements aren't specified in the task definition, the task uses the default Docker runtime.

The following shows the JSON format for the GPU requirements in a task definition:

```
{  
    "containerDefinitions": [  
        {  
            ...  
            "resourceRequirements" : [  
                {  
                    "type" : "GPU",  
                    "value" : "2"  
                }  
            ],  
        },  
        ...  
    }  
}
```

The following example demonstrates the syntax for a Docker container that specifies a GPU requirement. This container uses 2 GPUs, runs the `nvidia-smi` utility, and then exits.

```
{  
    "containerDefinitions": [  
        {  
            "memory": 80,  
            "essential": true,  
            "name": "gpu",  
            "image": "nvidia/cuda:11.0-base",  
            "resourceRequirements": [  
                {  
                    "type": "GPU",  
                    "value": "2"  
                }  
            ],  
            "command": [  
                "sh",  
                "-c",  
                "nvidia-smi"  
            ],  
            "cpu": 100  
        },  
        {  
            "family": "example-ecs-gpu"  
        }  
    ]  
}
```

Working with inference workloads on Amazon ECS

Amazon ECS supports machine learning inference workloads by enabling you to register [Amazon EC2 Inf1](#) instances to your clusters. Amazon EC2 Inf1 instances are powered by [AWS Inferentia](#) chips, which are custom built by AWS to provide high performance and lowest cost inference in the cloud. Machine learning models are deployed to containers using [AWS Neuron](#), a specialized software development kit (SDK) consisting of a compiler, runtime, and profiling tools that optimize the machine learning inference performance of Inferentia chips. AWS Neuron supports popular machine learning frameworks such as TensorFlow, PyTorch, and Apache MXNet (Incubating).

Considerations

Before you begin deploying Neuron on Amazon ECS, be aware of the following considerations:

- Your clusters can contain a mix of Inf1 and non-Inf1 instances.
- We recommend that you place only one task with an Inferentia resource requirement per Inf1 instance.
- When creating a service or running a standalone task, you can use instance type attributes when configuring task placement constraints to ensure which of your container instances the task is launched on. By doing this, you can more effectively use your resources while also ensuring your tasks for inference workloads land on your Inf1 instances. For more information, see [Amazon ECS task placement \(p. 513\)](#).

The following example runs a task on a Inf1.xlarge instance on your default cluster.

```
aws ecs run-task \
    --cluster default \
    --task-definition ecs-inference-task-def \
    --placement-constraints type=memberOf,expression="attribute:ecs.instance-type == Inf1.xlarge"
```

- Inferentia resource requirements can't be defined in a task definition. However, you can configure a container to use specific Inferentia available on the host container instance by using the `linuxParameters` parameter and specifying the device details. For more information, see [Task definition requirements \(p. 254\)](#).

Using the Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI

Amazon ECS provides an Amazon ECS-optimized AMI based on Amazon Linux 2 for Inferentia workloads that comes pre-configured with AWS Inferentia drivers and the AWS Neuron runtime for Docker. This AMI makes running machine learning workloads easier on Amazon ECS.

We recommend using the Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI when launching your Amazon EC2 Inf1 instances. You can retrieve the current Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI using the AWS CLI with the following command:

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

The following table provides a link to retrieve the current Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI IDs by Region.

Region name	Region	AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US East (Ohio)	us-east-2	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Milan)	eu-south-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

Task definition requirements

To deploy Neuron on Amazon ECS, your task definition must contain the container definition for a pre-built container serving the inference model for TensorFlow provided by AWS Deep Learning Containers. This container contains the AWS Neuron runtime and the TensorFlow Serving application. At start up, this container will fetch your model from Amazon S3, launch Neuron TensorFlow Serving with the saved model, and wait for prediction requests. The following container image has TensorFlow 1.15 and Ubuntu 18.04. A complete list of pre-built Deep Learning Containers optimized for Neuron is maintained on GitHub. For more information, see [Neuron Inference Containers](#).

763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.4-neuron-py37-ubuntu18.04

Alternatively, you can build your own Neuron sidecar container image. For more information, see [Tutorial: Neuron TensorFlow Serving](#) on GitHub.

Currently, the Inferentia resource requirement can't be defined in a task definition. However, you can configure a container to use specific Inferentia available on the host container instance using the `linuxParameters` parameter. The following is an example Linux containers on Fargate task definition, displaying the syntax to use.

```
{
    "family": "ecs-neuron",
    "executionRoleArn": "${YOUR_EXECUTION_ROLE}",
    "containerDefinitions": [
        {
            "entryPoint": [
                "/usr/local/bin/entrypoint.sh",
                "--port=8500",
                "--rest_api_port=9000",
                "--model_name=resnet50_neuron",
                "--model_base_path=s3://your-bucket-of-models/resnet50_neuron/"
            ],
            "portMappings": [
                {
                    "hostPort": 8500,
                    "protocol": "tcp",
                    "containerPort": 8500
                },
                {
                    "hostPort": 8501,
                    "protocol": "tcp",
                    "containerPort": 8501
                },
                {
                    "hostPort": 0,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "linuxParameters": {
                "devices": [
                    {
                        "containerPath": "/dev/neuron0",
                        "hostPath": "/dev/neuron0",
                        "permissions": [
                            "read",
                            "write"
                        ]
                    }
                ],
                "capabilities": {
                    "add": [
                        "IPC_LOCK"
                    ]
                }
            },
            "cpu": 0,
            "memoryReservation": 1000,
            "image": "763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.4-neuron-py37-ubuntu18.04",
            "essential": true,
            "name": "resnet50"
        }
    ]
}
```

Using data volumes in tasks

Amazon ECS 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, growing and shrinking automatically as you add and remove files. Your applications can have the storage they need, when they need it. Amazon EFS volumes are supported for tasks hosted on Fargate or Amazon EC2 instances. For more information, see [Amazon EFS volumes \(p. 257\)](#).
- FSx for Windows File Server volumes — Provides fully managed Microsoft Windows file servers, that are backed by a fully native Windows file system. When using FSx for Windows File Server together with Amazon ECS, you can provision your Windows tasks with persistent, distributed, shared, static file storage. For more information, see [FSx for Windows File Server volumes \(p. 260\)](#).

Windows containers on Fargate do not support this option.

- Docker volumes — A Docker-managed volume that is created under `/var/lib/docker/volumes` on the host Amazon EC2 instance. Docker volume drivers (also referred to as plugins) are used to integrate the volumes with external storage systems, such as Amazon EBS. The built-in `local` volume driver or a third-party volume driver can be used. Docker volumes are only supported when running tasks on Amazon EC2 instances. Windows containers only support the use of the `local` driver. To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see [Docker volumes \(p. 263\)](#).
- Bind mounts — A file or directory on the host, such as an Amazon EC2 instance or AWS Fargate, is mounted into a container. Bind mount host volumes are supported for tasks hosted on Fargate or Amazon EC2 instances. For more information, see [Bind mounts \(p. 268\)](#).

Topics

- [Fargate task storage \(p. 256\)](#)
- [Amazon EFS volumes \(p. 257\)](#)
- [FSx for Windows File Server volumes \(p. 260\)](#)
- [Docker volumes \(p. 263\)](#)
- [Bind mounts \(p. 268\)](#)

Fargate task storage

When provisioned, each Amazon ECS task hosted on AWS Fargate receives the following ephemeral storage for bind mounts. This can be mounted and shared among containers using the `volumes`, `mountPoints` and `volumesFrom` parameters in the task definition.

Fargate tasks using Windows platform version 1.0.0 or later

By default, Amazon ECS tasks hosted on Fargate using platform version 1.0.0 or later receive a minimum of 20 GiB of ephemeral storage. The total amount of ephemeral storage can be increased, up to a maximum of 200 GiB, by specifying the `ephemeralStorage` parameter in your task definition.

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

Fargate tasks using Linux platform version 1.4.0 or later

By default, Amazon ECS tasks 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, 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 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 owned encryption key.

Fargate tasks using Linux 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.

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, growing and shrinking automatically as you add and remove files. Your applications can have the storage they need, when they need it.

You can use Amazon EFS file systems with Amazon ECS to export file system data across your fleet of container instances. That way, your tasks have access to the same persistent storage, no matter the instance on which they land. However, you must configure your container instance AMI to mount the Amazon EFS file system before the Docker daemon starts. Also, your task definitions must reference volume mounts on the container instance to use the file system. The following sections help you get started using Amazon EFS with Amazon ECS.

For a tutorial, see [Tutorial: Using Amazon EFS file systems with Amazon ECS \(p. 790\)](#).

Amazon EFS volume considerations

The following should be considered when using Amazon EFS volumes:

- For tasks using the EC2 launch type, Amazon EFS file system support was added as a public preview with Amazon ECS-optimized AMI version 20191212 with container agent version 1.35.0. However, Amazon EFS file system support entered general availability with Amazon ECS-optimized AMI version 20200319 with container agent version 1.38.0, which contained the Amazon EFS access point and IAM authorization features. We recommend that you use Amazon ECS-optimized AMI version 20200319 or later to take advantage of these features. For more information, see [Amazon ECS-optimized AMI versions \(p. 342\)](#).

Note

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

Amazon Linux 2

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

Amazon Linux

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

- For tasks 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 \(p. 169\)](#).
- When using Amazon EFS volumes for tasks hosted on Fargate, Fargate creates a supervisor container that is responsible for managing the Amazon EFS volume. The supervisor container uses a small amount of the task's memory. The supervisor container is visible when querying the task metadata version 4 endpoint, but isn't visible in CloudWatch Container Insights. For more information, see [Task metadata endpoint version 4 \(p. 477\)](#).

Using Amazon EFS access points

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

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

Note

When creating an EFS access point, you 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 will enforce 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 easily provide secure access to specific datasets for your applications. For more information on using task IAM roles, see [IAM Roles for Tasks \(p. 699\)](#).

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 will be used. Specifying / has the same effect as omitting this parameter.

Important

If an EFS access point is specified in the authorizationConfig, the root directory parameter must either be omitted or set to /, which enforces 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 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 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](#).

FSx for Windows File Server volumes

FSx for Windows File Server provides fully managed Microsoft Windows file servers, that are backed by a fully native Windows file system. When using FSx for Windows File Server together with ECS, you can provision your Windows tasks with persistent, distributed, shared, static file storage. For more information, see [What Is FSx for Windows File Server?](#).

Note

EC2 instances that use the Amazon ECS-Optimized Windows Server 2016 Full AMI, do not support FSx for Windows File Server ECS task volumes.

You cannot use FSx for Windows File Server volumes in a Windows containers on Fargate configuration.

You can use FSx for Windows File Server to deploy Windows workloads that require access to shared external storage, highly-available regional storage, or high-throughput storage. You can mount one or more FSx for Windows File Server file system volumes to an ECS container running on an ECS Windows instance. You can share FSx for Windows File Server file system volumes between multiple ECS containers within a single ECS task.

To enable the use of FSx for Windows File Server with ECS, you need to include the FSx for Windows File Server file system id and related information in a task definition as shown in the following example task definition JSON snippet. Before creating and running a task definition, you need the following.

- An ECS Windows EC2 instance that is joined to a valid domain, hosted by an [AWS Directory Service for Microsoft Active Directory](#), On-premises Active Directory or self-hosted Active Directory on Amazon EC2.
- An AWS Secrets Manager secret or Systems Manager parameter that contains the credentials that are used to domain join the Active Directory and attach the FSx for Windows File Server file system. The credential values are the name and password credentials that you entered when creating the Active Directory.

The following sections are provided to help you get started using FSx for Windows File Server with Amazon ECS.

For a tutorial, see [Tutorial: Using FSx for Windows File Server file systems with Amazon ECS \(p. 796\)](#).

FSx for Windows File Server volume considerations

Consider the following when using FSx for Windows File Server volumes.

- FSx for Windows File Server with Amazon ECS only supports Windows Amazon EC2 instances. Linux Amazon EC2 instances aren't supported.
- Amazon ECS only supports FSx for Windows File Server. Amazon ECS doesn't support FSx for Lustre.
- FSx for Windows File Server with Amazon ECS doesn't support AWS Fargate.
- FSx for Windows File Server with Amazon ECS with awsvpc network mode requires version 1.54.0 or later of the container agent.
- The maximum number of drive letters that can be used for an Amazon ECS task is 23. Each task with an FSx for Windows File Server volume gets a drive letter assigned to it.
- Task resource cleanup time is 3 hours by default. A file mapping created by a task persists for 3 hours even if no tasks are using it. The default cleanup time can be configured by using the Amazon ECS environment variable `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).
- Tasks typically only run in the same VPC as the FSx for Windows File Server file system. However, it is possible to have cross-VPC support if there is an established network connectivity between the Amazon ECS cluster VPC and the FSx for Windows File Server file-system through VPC peering.
- You control access to an FSx for Windows File Server file system at the network level by configuring the VPC security groups. Only tasks hosted on EC2 instances joined to the AD domain with correctly configured AD security groups will be able to access the FSx for Windows File Server file-share. If the security groups are misconfigured, ECS will fail the Task launch with the following error message: `unable to mount file system fs-id`.
- FSx for Windows File Server is integrated with AWS Identity and Access Management (IAM) to control the actions that your IAM users and groups can take on specific FSx for Windows File Server resources. With client authorization, customers can define IAM roles that allow or deny access to specific FSx for Windows File Server file systems, optionally require read-only access, and optionally allow or disallow root access to the file system from the client. For more information, see [Security](#) in the Amazon FSx Windows User Guide.

Specifying an FSx for Windows File Server file system in your task definition

To use FSx for Windows File Server 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-fsx",
    "image": "iis:2",
    "entryPoint": [
        "sh",
        "-c"
    ],
    "command": [
        "ls -la \\mount\\fsx"
    ],
    "mountPoints": [
        {
            "sourceVolume": "myFsxVolume",
            "containerPath": "\\mount\\fsx",
            "readOnly": true
        }
    ]
},
"volumes": [
    {
        "name": "myFsxVolume",
        "FSxWindowsFileServerVolumeConfiguration": {
            "fileSystemId": "fs-1234",
            "rootDirectory": "\\path\\to\\my\\data",
            "credentialsParameter": "arn-1234",
            "domain": "corp.fullyqualified.com",
        }
    }
]
}
```

`FSxWindowsFileServerVolumeConfiguration`

Type: Object

Required: No

This parameter is specified when you are using [FSx for Windows File Server](#) file system for task storage.

`fileSystemId`

Type: String

Required: Yes

The FSx for Windows File Server file system ID to use.

`rootDirectory`

Type: String

Required: Yes

The directory within the FSx for Windows File Server file system to mount as the root directory inside the host.

`authorizationConfig`

`credentialsParameter`

Type: String

Required: Yes

The authorization credential options:

- Amazon Resource Name (ARN) of an [Secrets Manager](#) secret.
- Amazon Resource Name (ARN) of an [Systems Manager](#) parameter.

domain

Type: String

Required: Yes

A fully qualified domain name hosted by an [AWS Directory Service](#) Managed Microsoft AD (Active Directory) or self-hosted EC2 AD.

Credential storage methods

There are two different methods of storing credentials for use with the credentials parameter.

- **AWS Secrets Manager secret**

This credential can be created in the AWS Secrets Manager console by using the *Other type of secret* category. You add a row for each key/value pair, username/admin and password/*password*.

- **Systems Manager parameter**

This credential can be created in the Systems Manager parameter console by entering text in the form shown in the following example code snippet.

```
{  
    "username": "admin",  
    "password": "password"  
}
```

The `credentialsParameter` in the task definition `FSxWindowsFileServerVolumeConfiguration` parameter will hold either the secret ARN or the Systems Manager parameter ARN. For more information, see [What is AWS Secrets Manager](#) in the *Secrets Manager User Guide* and [Systems Manager Parameter Store](#) from the *Systems Manager User Guide*.

Docker volumes

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

To use Docker volumes, specify a `dockerVolumeConfiguration` in your task definition. For more information, see [Using Volumes](#).

Some common use cases for Docker volumes are:

- To provide persistent data volumes for use with containers
- To share a defined data volume at different locations on different containers on the same container instance
- To define an empty, nonpersistent data volume and mount it on multiple containers within the same task
- To provide a data volume to your task that is managed by a third-party driver

Docker volume considerations

The following should be considered when using Docker volumes:

- Docker volumes are only supported when using the EC2 launch type.
- Windows containers only support the use of the local driver.
- If a third-party driver is used, it should be installed and active on the container instance prior to the container agent starting. If the third-party driver isn't active prior to the agent starting, you can restart the container agent using one of the following commands:

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI:

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

Specifying a Docker volume in your task definition

Before your containers can use data volumes, you must specify the volume and mount point configurations in your task definition. This section describes the volume configuration for a container. For tasks that use a Docker volume, specify a `dockerVolumeConfiguration`. For tasks that use a bind mount host volume, specify a `host` and optional `sourcePath`.

The task definition JSON shown below shows the syntax for the `volumes` and `mountPoints` objects for a container.

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

`name`

Type: String

Required: No

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

`dockerVolumeConfiguration`

Type: Object

Required: No

This parameter is specified when using Docker volumes. Docker volumes are only supported when using the EC2 launch type. Windows containers only support the use of the `local` driver. To use bind mounts, specify a host instead.

`scope`

Type: String

Valid Values: `task` | `shared`

Required: No

The scope for the Docker volume, which determines its lifecycle. Docker volumes that are scoped to a task are automatically provisioned when the task starts destroyed when the task is cleaned up. Docker volumes that are scoped as shared persist after the task stops.

`autoprovision`

Type: Boolean

Default value: `false`

Required: No

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

`driver`

Type: String

Required: No

The Docker volume driver to use. The driver value must match the driver name provided by Docker because it is used for task placement. If the driver was installed using the Docker plugin CLI, use `docker plugin ls` to retrieve the driver name from your container instance. If the driver was installed using another method, use Docker plugin discovery to retrieve the driver name. For more information, see [Docker plugin discovery](#). This parameter maps to `Driver` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--driver` option to `docker volume create`.

`driverOpts`

Type: String

Required: No

A map of Docker driver specific options to pass through. This parameter maps to `DriverOpts` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--opt` option to `docker volume create`.

`labels`

Type: String

Required: No

Custom metadata to add to your Docker volume. This parameter maps to `Labels` in the [Create a volume](#) section of the [Docker Remote API](#) and the `--label` option to [docker volume create](#).

`mountPoints`

Type: Object Array

Required: No

The mount points for data volumes in your container.

This parameter maps to `Volumes` in the [Create a container](#) section of the [Docker Remote API](#) and the `--volume` option to [docker run](#).

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives.

`sourceVolume`

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

`containerPath`

Type: String

Required: Yes, when `mountPoints` are used

The path on the container to mount the volume at.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

Examples

The following are examples showing the use of Docker volumes.

To provide nonpersistent storage for a container using a Docker volume

In this example, you want a container to use an empty data volume that you aren't interested in keeping after the task has finished. For example, you may have a container that needs to access some scratch file storage location during a task. This task can be achieved using a Docker volume.

1. In the task definition `volumes` section, define a data volume with `name` and `DockerVolumeConfiguration` values. In this example, we specify the scope as `task` so the volume is deleted after the task stops and use the built-in `local` driver.

```
"volumes": [
  {
    "name": "scratch",
    "dockerVolumeConfiguration" : {
      "scope": "task",
      "driver": "local",
      "labels": {
```

```

        "scratch": "space"
    }
}
]
```

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

```

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

To provide persistent storage for a container using a Docker volume

In this example, you want a shared volume for multiple containers to use and you want it to persist after any single task using it has stopped. The built-in `local` driver is being used so the volume is still tied to the lifecycle of the container instance.

1. In the task definition `volumes` section, define a data volume with `name` and `DockerVolumeConfiguration` values. In this example, specify a `shared` scope so the volume persists, set `autoprovision` to `true` so that the volume is created for use, and use the built-in `local` driver.

```

"volumes": [
{
    "name": "database",
    "dockerVolumeConfiguration" : {
        "scope": "shared",
        "autoprovision": true,
        "driver": "local",
        "labels": {
            "database": "database_name"
        }
    }
}]
```

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

```

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

```
{
  "name": "container-2",
  "mountPoints": [
    {
      "sourceVolume": "database",
      "containerPath": "/var/database"
    }
  ]
}
```

Bind mounts

With bind mounts, a file or directory on a host, such as an Amazon EC2 instance or AWS Fargate, is mounted into a container. Bind mounts are supported for tasks hosted on both Fargate and Amazon EC2 instances. By default, bind mounts are tied to the lifecycle of the container using them. Once all containers using a bind mount are stopped, for example when a task is stopped, the data is removed. For tasks hosted on Amazon EC2 instances, the data can be tied to the lifecycle of the host Amazon EC2 instance by specifying a host and optional `sourcePath` value in your task definition. For more information, see [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 mount a host data volume in one or more containers.
- To share a data volume from a source container with other containers in the same task.
- To expose a path and its contents from a Dockerfile to one or more containers.

Considerations when using bind mounts

When using bind mounts, the following should be considered.

- For tasks 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 specified in the `VOLUME` directive is the same as the `containerPath` 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 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`. These permissions can be customized in the Dockerfile. The following example defines the owner of the directory as `node`.

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

```
USER node
VOLUME ["/var/log/exported"]
```

- For tasks hosted on Amazon EC2 instances, when a host and sourcePath value are not specified, the Docker daemon manages the bind mount for you. When no containers reference this bind mount, the Amazon ECS container agent task cleanup service eventually deletes it (by default, this happens 3 hours after the container exits, but you can configure this duration with the `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` agent variable). For more information, see [Amazon ECS container agent configuration \(p. 454\)](#). If you need this data to persist beyond the lifecycle of the container, specify a sourcePath value for the bind mount.

Specifying a bind mount in your task definition

For Amazon ECS tasks hosted on either Fargate or Amazon EC2 instances, the following task definition JSON snippet shows the syntax for the volumes, mountPoints, and ephemeralStorage objects for a task definition.

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

For Amazon ECS tasks hosted on Amazon EC2 instances, you can use the optional host parameter and a sourcePath when specifying the task volume details, which when specified ties the bind mount to the lifecycle of the task rather than the container.

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

The following describes each task definition parameter in more detail.

name

Type: String

Required: No

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

`host`

Required: No

This parameter is specified when using bind mounts. To use Docker volumes, specify a `dockerVolumeConfiguration` instead. The contents of the `host` parameter determine whether your bind mount data volume persists on the host container instance and where it is stored. If the `host` parameter is empty, then the Docker daemon assigns a host path for your data volume, but the data is not guaranteed to persist after the containers associated with it stop running.

Bind mount host volumes are supported when using either the EC2 or Fargate launch types.

Windows containers can mount whole directories on the same drive as `$env:ProgramData`.
`sourcePath`

Type: String

Required: No

When the `host` parameter is used, specify a `sourcePath` to declare the path on the host container instance that is presented to the container. If this parameter is empty, then the Docker daemon has assigned a host path for you. If the `host` parameter contains a `sourcePath` file location, then the data volume persists at the specified location on the host container instance until you delete it manually. If the `sourcePath` value does not exist on the host container instance, the Docker daemon creates it. If the location does exist, the contents of the source path folder are exported.

`mountPoints`

Type: Object Array

Required: No

The mount points for data volumes in your container.

This parameter maps to `Volumes` in the [Create a container](#) section of the [Docker Remote API](#) and the `--volume` option to [`docker run`](#).

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers cannot mount directories on a different drive, and mount point cannot be across drives.

`sourceVolume`

Type: String

Required: Yes, when `mountPoints` are used

The name of the volume to mount.

`containerPath`

Type: String

Required: Yes, when `mountPoints` are used

The path on the container to mount the volume at.

`readOnly`

Type: Boolean

Required: No

If this value is `true`, the container has read-only access to the volume. If this value is `false`, then the container can write to the volume. The default value is `false`.

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 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 may 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 so that they mount the volume.

```
"containerDefinitions": [  
    {  
        "name": "database1",  
        "image": "my-repo/database",  
        "cpu": 100,  
        "memory": 100,
```

```

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

```

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

In this example, you have a Dockerfile that writes data that you want to mount inside a container. This example works for tasks hosted on Fargate or Amazon EC2 instances.

1. Create a Dockerfile. The following example uses the public Amazon Linux 2 container image and creates a file 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 so they mount the storage. The `containerPath` value must match the absolute path specified in the `VOLUME` directive from the Dockerfile.

```

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

```

To provide an empty data volume for a container that is tied to the lifecycle of the host Amazon EC2 instance

For tasks hosted on Amazon EC2 instances, you can use bind mounts and have the data tied to the lifecycle of the host Amazon EC2 instance rather than the containers referencing the volume. This is done by using the `host` parameter and specifying a `sourcePath` value. Any files that exist at the `sourcePath` are presented to the containers at the `containerPath` value, and any files that are written to the `containerPath` value are written to the `sourcePath` value on the host Amazon EC2 instance.

Important

Amazon ECS doesn't sync your storage across Amazon EC2 instances. Tasks that use persistent storage can be placed on any Amazon EC2 instance in your cluster that has available capacity. If your tasks require persistent storage after stopping and restarting, you should always specify the same Amazon EC2 instance at task launch time with the AWS CLI [start-task](#) command. You can also use Amazon EFS volumes for persistent storage. For more information, see [Amazon EFS volumes \(p. 257\)](#).

1. In the task definition `volumes` section, define a bind mount with `name` and `sourcePath` values. In the following example, the host Amazon EC2 instance contains data at `/ecs/webdata` that you want to mount inside the container.

```

"volumes": [
  {
    "name": "webdata",
    "host": {
      "sourcePath": "/ecs/webdata"
    }
  }
]

```

2. In the `containerDefinitions` section, define a container with a `mountPoints` value that references the name of the bind mount and the `containerPath` value to mount the bind mount at on the container.

```
"containerDefinitions": [
  {
    "name": "web",
    "image": "nginx",
    "cpu": 99,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webdata",
        "containerPath": "/usr/share/nginx/html"
      }
    ]
  }
]
```

To mount a defined volume on multiple containers at different locations

You can define a data volume in a task definition and mount that volume at different locations on different containers. For example, your host container has a website data folder at `/data/webroot`, and you may want to mount that data volume as read-only on two different web servers that have different document roots.

1. In the task definition `volumes` section, define a data volume with the name `webroot` and the source path `/data/webroot`.

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

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

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

```

    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
        "containerPath": "/var/www/html",
        "readOnly": true
      }
    ]
  },
  {
    "name": "web-server-2",
    "image": "my-repo/sles11-apache",
    "cpu": 100,
    "memory": 100,
    "portMappings": [
      {
        "containerPort": 8080,
        "hostPort": 8080
      }
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "webroot",
        "containerPath": "/srv/www/htdocs",
        "readOnly": true
      }
    ]
  }
]

```

To mount volumes from another container using `volumesFrom`

You can define one or more volumes on a container, and then use the `volumesFrom` parameter in a different container definition (within the same task) to mount all of the volumes from the `sourceContainer` at their originally defined mount points. The `volumesFrom` parameter applies to volumes defined in the task definition, and those that are built into the image with a Dockerfile.

1. (Optional) To share a volume that is built into an image, you need to build the image with the volume declared in a `VOLUME` instruction. The following example Dockerfile uses an `httpd` image and then adds a volume and mounts it at `dockerfile_volume` in the Apache document root (which is the folder used by the `httpd` web server):

```

FROM httpd
VOLUME ["/usr/local/apache2/htdocs/dockerfile_volume"]

```

You can build an image with this Dockerfile and push it to a repository, such as Docker Hub, and use it in your task definition. The example `my-repo/httpd_dockerfile_volume` image used in the following steps was built with the above Dockerfile.

2. Create a task definition that defines your other volumes and mount points for the containers. In this example `volumes` section, you create an empty volume called `empty`, which the Docker daemon manages. There is also a host volume defined called `host_etc`, which exports the `/etc` folder on the host container instance.

```

{
  "family": "test-volumes-from",
  "volumes": [
    {
      "name": "empty",
      "host": {}
    }
  ]
}

```

```

},
{
  "name": "host_etc",
  "host": {
    "sourcePath": "/etc"
  }
},
]
,
```

In the container definitions section, create a container that mounts the volumes defined earlier. In this example, the web container (which uses the image built with a volume in the Dockerfile) mounts the empty and host_etc volumes.

```

"containerDefinitions": [
  {
    "name": "web",
    "image": "my-repo/httpd\_dockerfile\_volume",
    "cpu": 100,
    "memory": 500,
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 80
      }
    ],
    "mountPoints": [
      {
        "sourceVolume": "empty",
        "containerPath": "/usr/local/apache2/htdocs/empty_volume"
      },
      {
        "sourceVolume": "host_etc",
        "containerPath": "/usr/local/apache2/htdocs/host_etc"
      }
    ],
    "essential": true
  },
]
```

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

```

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

```
        "essential": false
    }
}
}
```

When this task is run, the two containers mount the volumes, and the `command` in the `busybox` container writes the date and time to a file called `date` in each of the volume folders. The folders are then visible at the website displayed by the web container.

Note

Because the `busybox` container runs a quick command and then exits, it must be set as `"essential": false` in the container definition. Otherwise, it stops the entire task when it exits.

Managing container swap space

Amazon ECS enables you to control the usage of swap memory space on your Linux container instances at the container level. Using a per-container swap configuration, each container within a task definition can have swap enabled or disabled, and for those that have it enabled, the maximum amount of swap space used can be limited. For example, latency-critical containers can have swap disabled, whereas containers with high transient memory demands can have swap turned on to reduce the chances of out-of-memory errors when the container is under load.

The swap configuration for a container is managed by the following container definition parameters:

`maxSwap`

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

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

`swappiness`

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

The following is an example showing the JSON syntax:

```
"containerDefinitions": [{

    ...
    "linuxParameters": {
        "maxSwap": integer,
        "swappiness": integer
    },
    ...
}]
```

Container swap considerations

Consider the following when you use a per-container swap configuration.

- Swap space must be enabled and allocated on the container instance for the containers to use.

Note

The Amazon ECS-optimized AMIs do not have swap enabled by default. You must enable swap on the instance to use this feature. For more information, see [Instance Store Swap Volumes in the Amazon EC2 User Guide for Linux Instances](#) or [How do I allocate memory to work as swap space in an Amazon EC2 instance by using a swap file?](#).

- The swap space container definition parameters are only supported for task definitions using the EC2 launch type.
- This feature is only supported for Linux containers.
- If the `maxSwap` and `swappiness` container definition parameters are omitted from a task definition, each container will have a default `swappiness` value of 60 and the total swap usage will be limited to two times the memory reservation of the container.

Amazon ECS task networking

Important

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Fargate task networking](#) in the [Amazon Elastic Container Service User Guide for AWS Fargate](#).

The networking behavior of Amazon ECS tasks hosted on Amazon EC2 instances is dependent on the *network mode* defined in the task definition. The following are the available network modes. Amazon ECS recommends using the `awsvpc` network mode unless you have a specific need to use a different network mode.

- `awsvpc` — The task is allocated its own elastic network interface (ENI) and a primary private IPv4 address. This gives the task the same networking properties as Amazon EC2 instances.
- `bridge` — The task utilizes Docker's built-in virtual network which runs inside each Amazon EC2 instance hosting the task.
- `host` — The task bypasses Docker's built-in virtual network and maps container ports directly to the ENI of the Amazon EC2 instance hosting the task. As a result, you can't run multiple instantiations of the same task on a single Amazon EC2 instance when port mappings are used.
- `none` — The task has no external network connectivity.

For more information about Docker networking, see [Networking overview](#).

Topics

- [Task networking with the awsvpc network mode \(p. 278\)](#)
- [Task networking with the bridge network mode \(p. 282\)](#)
- [Task networking with the host network mode \(p. 283\)](#)

Task networking with the awsvpc network mode

The task networking features provided by the `awsvpc` network mode give Amazon ECS tasks the same networking properties as Amazon EC2 instances. Using the `awsvpc` network mode simplifies container networking and gives you more control over how containerized applications communicate with each

other and other services within your VPCs. The `awsvpc` network mode also provides greater security for your containers by enabling you to use security groups and network monitoring tools at a more granular level within your tasks. Because each task gets its own elastic network interface (ENI), you can also take advantage of other Amazon EC2 networking features like VPC Flow Logs so that you can monitor traffic to and from your tasks. Additionally, containers that belong to the same task can communicate over the `localhost` interface.

The task ENI is fully managed by Amazon ECS. Amazon ECS creates the ENI and attaches it to the host Amazon EC2 instance with the specified security group. The task sends and receives network traffic over the ENI in the same way that Amazon EC2 instances do with their primary network interfaces. Each task ENI is assigned a private IPv4 address by default. If your VPC is enabled for dual-stack mode and you use a subnet with an IPv6 CIDR block, the task ENI will also receive an IPv6 address. Each task can only have one ENI.

These ENIs are visible in the Amazon EC2 console for your account, but they cannot be detached manually or modified by your account. This is to prevent accidental deletion of an ENI that is associated with a running task. You can view the ENI attachment information for tasks in the Amazon ECS console or with the [DescribeTasks](#) API operation. When the task stops or if the service is scaled down, the task ENI is detached and deleted.

If your account, IAM user, or role has opted in to the `awsvpcTrunking` account setting and you have launched a container instance with the increased ENI density, Amazon ECS also creates and attaches a "trunk" network interface for your container instance. The trunk network is fully managed by Amazon ECS. The trunk ENI is deleted when you either terminate or deregister your container instance from the Amazon ECS cluster. For more information on opting in to the `awsvpcTrunking` account setting, see [Working with container instances with increased ENI limits \(p. 373\)](#).

Considerations

There are several things to consider when using the `awsvpc` network mode.

Linux considerations

The following are considerations when you use the Linux operating system:

- Tasks and services that use the `awsvpc` network mode require the Amazon ECS service-linked role to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster, or if you create or update a service, in the AWS Management Console. For more information, see [Service-linked role for Amazon ECS \(p. 685\)](#). You can also create the service-linked role with the following AWS CLI command:

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

- Your Amazon EC2 Linux instance requires version 1.15.0 or later of the container agent to run tasks that use the `awsvpc` network mode. If you are using an Amazon ECS-optimized AMI, your instance needs at least version 1.15.0-4 of the `ecs-init` package as well.
- Amazon ECS populates the hostname of the task with an Amazon-provided (internal) DNS hostname when both the `enableDnsHostnames` and `enableDnsSupport` options are enabled on your VPC. If these options are not enabled, the DNS hostname of the task will be a random hostname. For more information on the DNS settings for a VPC, see [Using DNS with Your VPC](#) in the *Amazon VPC User Guide*.
- Each Amazon ECS task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the Amazon EC2 instance that hosts it. There is a default limit to the number of network interfaces that can be attached to an Amazon EC2 Linux instance, and the primary network interface counts as one. For example, by default a `c5.large` instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs to the instance. Because each task using the `awsvpc` network mode requires

an ENI, you can typically only run two such tasks on this instance type. For more information on the default ENI limits for each instance type, see [IP addresses per network interface per instance type](#) in the *Amazon EC2 User Guide for Linux Instances*.

- Amazon ECS supports the launch of Amazon EC2 Linux instances using supported instance types with increased ENI density. When you opt in to the `awsvpcTrunking` account setting and register Amazon EC2 Linux instances using these instance types to your cluster, these instances have higher ENI limits. This enables you to place more tasks on each Amazon EC2 Linux instance. To take advantage of the increased ENI density with the trunking feature, your Amazon EC2 instance requires at least version 1.28.1 of the container agent. If you are using an Amazon ECS-optimized AMI, your instance also requires at least version 1.28.1-2 of the `ecs-init` package. For more information on opting in to the `awsvpcTrunking` account setting, see [Account settings \(p. 324\)](#). For more information on ENI trunking, see [Elastic network interface trunking \(p. 372\)](#).
- When hosting tasks that use the `awsvpc` network mode on Amazon EC2 Linux instances, your task ENIs are not given public IP addresses. To access the internet, tasks should be launched in a private subnet that is configured to use a NAT gateway. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide*. Inbound network access must be from within the VPC using the private IP address or routed through a load balancer from within the VPC. Tasks launched within public subnets do not have access to the internet.
- Amazon ECS only accounts for the ENIs that it attaches to your Amazon EC2 Linux instances for you. If you have attached ENIs to your instances manually, then Amazon ECS could try to place a task on an instance with too few available network adapter attachments. In this case, the task would time out, move from `PROVISIONING` to `DEPROVISIONING`, and then to `STOPPED`. We recommend that you do not attach ENIs to your instances manually.
- Amazon EC2 Linux instances must be registered with the `ecs.capability.task-eni` capability to be considered for placement of tasks with the `awsvpc` network mode. Instances running version 1.15.0-4 or later of `ecs-init` are registered with this attribute automatically.
- The ENIs that are created and attached to your Amazon EC2 Linux instances cannot be detached manually or modified by your account. This is to prevent the accidental deletion of an ENI that is associated with a running task. To release the ENIs for a task, stop the task.
- There is a limit of 16 subnets and 5 security groups that are able to be specified in the `awsvpcConfiguration` when running a task or creating a service that uses the `awsvpc` network mode. For more information, see [AwsVpcConfiguration](#) in the *Amazon Elastic Container Service API Reference*.
- When a task is started with the `awsvpc` network mode, the Amazon ECS container agent creates an additional pause container for each task before starting the containers in the task definition. It then configures the network namespace of the pause container by running the `amazon-ecs-cni-plugins` CNI plugins. The agent then starts the rest of the containers in the task so that they share the network stack of the pause container. This means that all containers in a task are addressable by the IP addresses of the ENI, and they can communicate with each other over the `localhost` interface.
- Services with tasks that use the `awsvpc` network mode only support Application Load Balancers and Network Load Balancers; Classic Load Balancers are not supported. Also, when you create any target groups for these services, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an ENI, not with an Amazon EC2 Linux instance. For more information, see [Service load balancing \(p. 574\)](#).
- If a VPC is updated, for example to change the DHCP options set it uses, and you want tasks using the VPC to pick up the changes, those tasks must be stopped and new tasks started.

Windows considerations

The following are considerations when you use the Windows operating system:

- Container instances using the Amazon ECS-optimized Windows Server 2016 AMI can't host tasks that use the `awsvpc` network mode. If you have a cluster that contains Amazon ECS-optimized Windows Server 2016 AMIs and Windows AMIs that do support `awsvpc` network mode, tasks that use `awsvpc`

network mode are not launched on the Windows 2016 Server instances, but will be launched on instances that support awsvpc network mode.

- Currently, CloudWatch metrics are not supported for Windows containers that use the awsvpc network mode.
- Tasks and services that use the awsvpc network mode require the Amazon ECS service-linked role to provide Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster, or if you create or update a service, in the AWS Management Console. For more information, see [Service-linked role for Amazon ECS \(p. 685\)](#). You can also create the service-linked role with the following AWS CLI command:

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

- Your Amazon EC2 Windows instance requires version 1.54.0 or later of the container agent to run tasks that use the awsvpc network mode. When you bootstrap the instance, you must configure the options that are required for awsvpc network mode. For more information, see [the section called "Bootstrap Container Instances" \(p. 411\)](#).
- Amazon ECS populates the hostname of the task with an Amazon-provided (internal) DNS hostname when both the enableDnsHostnames and enableDnsSupport options are enabled on your VPC. If these options are not enabled, the DNS hostname of the task will be a random hostname. For more information on the DNS settings for a VPC, see [Using DNS with Your VPC](#) in the *Amazon VPC User Guide*.
- Each Amazon ECS task that uses the awsvpc network mode receives its own elastic network interface (ENI), which is attached to the Amazon EC2 Windows instance that hosts it. There is a default limit to the number of network interfaces that can be attached to an Amazon EC2 Windows instance, and the primary network interface counts as one. For example, by default a c5.large instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs to the instance. Because each task using the awsvpc network mode requires an ENI, you can typically only run two such tasks on this instance type. For more information on the default ENI limits for each instance type, see [IP addresses per network interface per instance type](#) in the *Amazon EC2 User Guide for Windows Instances*.
- When hosting tasks that use the awsvpc network mode on Amazon EC2 Windows instances, your task ENIs are not given public IP addresses. To access the internet, tasks should be launched in a private subnet that is configured to use a NAT gateway. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide*. Inbound network access must be from within the VPC using the private IP address or routed through a load balancer from within the VPC. Tasks launched within public subnets do not have access to the internet.
- Amazon ECS only accounts for the ENIs that it attaches to your Amazon EC2 Windows instances for you. If you have attached ENIs to your instances manually, then Amazon ECS could try to place a task on an instance with too few available network adapter attachments. In this case, the task would time out, move from PROVISIONING to DEPROVISIONING, and then to STOPPED. We recommend that you do not attach ENIs to your instances manually.
- Amazon EC2 Windows instances must be registered with the `ecs.capability.task-eni` capability to be considered for placement of tasks with the awsvpc network mode.
- The ENIs that are created and attached to your Amazon EC2 Windows instances cannot be detached manually or modified by your account. This is to prevent the accidental deletion of an ENI that is associated with a running task. To release the ENIs for a task, stop the task.
- There is a limit of 16 subnets and 5 security groups that are able to be specified in the `awsvpcConfiguration` when running a task or creating a service that uses the awsvpc network mode. For more information, see [AwsVpcConfiguration](#) in the *Amazon Elastic Container Service API Reference*.
- When a task is started with the awsvpc network mode, the Amazon ECS container agent creates an additional pause container for each task before starting the containers in the task definition. It then configures the network namespace of the pause container by running the `amazon-ecs-cni-plugins` CNI plugins. The agent then starts the rest of the containers in the task so that they share the

network stack of the pause container. This means that all containers in a task are addressable by the IP addresses of the ENI, and they can communicate with each other over the `localhost` interface.

- Services with tasks that use the `awsvpc` network mode only support Application Load Balancers and Network Load Balancers; Classic Load Balancers are not supported. Also, when you create any target groups for these services, you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an ENI, not with an Amazon EC2 Windows instance. For more information, see [Service load balancing \(p. 574\)](#).
- If a VPC is updated, for example to change the DHCP options set it uses, and you want tasks using the VPC to pick up the changes, those tasks must be stopped and new tasks started.
- The following are not supported when you use `awsvpc` network mode in an EC2 Windows configuration:
 - Dual-stack configuration
 - IPv6
 - ENI trunking

Enabling task networking

In order for tasks to use the `awsvpc` network mode, it must be specified in the task definition. For more information, see [Network mode \(p. 211\)](#). Then, when you run a task or create a service, specify a network configuration that includes one or more subnets in which to place your tasks and one or more security groups to attach to its associated ENI. The tasks are placed on compatible Amazon EC2 instances in the same Availability Zones as those subnets, and the specified security groups are associated with the ENI that is provisioned for the task.

Using 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*.

One of the benefits of using a VPC in dual-stack mode is that tasks that are assigned an IPv6 address are able to access the internet as long as the VPC is configured with either an internet gateway or an egress-only internet gateway. NAT gateways are not needed. For more information, see [Internet gateways](#) and [Egress-only internet gateways](#) in the *Amazon VPC User Guide*.

Amazon ECS tasks are assigned an IPv6 address if the following conditions are met:

- The Amazon EC2 Linux instance hosting the task is using version 1.45.0 or later of the container agent. For information on checking the agent version your instance is using, and updating it if needed, see [Updating the Amazon ECS container agent \(p. 448\)](#).
- The `dualStackIPv6` account setting is enabled. For more information, see [Account settings \(p. 324\)](#).
- Your task is using the `awsvpc` network mode.
- Your VPC and subnet are configured for IPv6 and that network interfaces created in the specified subnet should be assigned an IPv6 address. For more information about configuring your VPC for dual-stack mode, see [Migrating to IPv6](#) and [Modify the IPv6 addressing attribute for your subnet](#) in the *Amazon VPC User Guide*.

Task networking with the bridge network mode

Amazon ECS tasks using the `bridge` network mode utilize Docker's built-in virtual network which runs inside each container. The bridge is an internal network namespace that allows each container connected

to the same bridge network to communicate with each other. It also provides an isolation boundary from containers that aren't connected to the same bridge network.

With the bridge network mode, you use static or dynamic port mappings to map ports in the container with ports on the Amazon EC2 host. For more information, see [Choosing a network mode](#) in the *Amazon ECS Best Practices Guide*.

Task networking with the host network mode

With the host network mode, the networking of the container is tied directly to the Amazon EC2 instance host that the container is running on. Each container will receive traffic over the IP address of the Amazon EC2 instance hosting it. For more information, see [Choosing a network mode](#) in the *Amazon ECS Best Practices Guide*.

Using the awslogs log driver

You can configure the containers in your tasks to send log information to CloudWatch Logs. If you are using the Fargate launch type for your tasks, this allows you to view the logs from your containers. If you are using the EC2 launch type, this enables you to view different logs from your containers in one convenient location, and it prevents your container logs from taking up disk space on your container instances. This topic helps you get started using the awslogs log driver in your task definitions.

Note

The type of information that is logged by the containers in your task depends mostly on their `ENTRYPOINT` command. By default, the logs that are captured show the command output that you would normally see in an interactive terminal if you ran the container locally, which are the `STDOUT` and `STDERR` I/O streams. The awslogs log driver simply passes these logs from Docker to CloudWatch Logs. For more information on how Docker logs are processed, including alternative ways to capture different file data or streams, see [View logs for a container or service](#) in the Docker documentation.

To send system logs from your Amazon ECS container instances to CloudWatch Logs, see [Monitoring your container instances \(p. 426\)](#). For more information about CloudWatch Logs, see [Monitoring Log Files](#) and [CloudWatch Logs quotas](#) in the *Amazon CloudWatch Logs User Guide*.

Enabling the awslogs log driver for your containers

If you are using the Fargate launch type for your tasks, all you need to do to enable the awslogs log driver is add the required `logConfiguration` parameters to your task definition. For more information, see [Specifying a log configuration in your task definition \(p. 286\)](#).

If you are using the EC2 launch type for your tasks and want to enable the awslogs log driver, your Amazon ECS container instances require at least version 1.9.0 of the container agent. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

Note

If you are not using the Amazon ECS-optimized AMI (with at least version 1.9.0-1 of the `ecs-init` package) for your container instances, you also need to specify that the awslogs logging driver is available on the container instance when you start the agent by using the following environment variable in your `docker run` statement or environment variable file. For more information, see [Installing the Amazon ECS container agent \(p. 431\)](#).

```
ECS_AVAILABLE_LOGGING_DRIVERS='["json-file", "awslogs"]'
```

Your Amazon ECS container instances also require `logs:CreateLogStream` and `logs:PutLogEvents` permission on the IAM role with which you launch your container instances. If you created your Amazon

ECS container instance role before `awslogs` log driver support was enabled in Amazon ECS, then you might need to add this permission. If your container instances use the managed IAM policy for container instances, then your container instances should have the correct permissions. For information about checking your Amazon ECS container instance role and attaching the managed IAM policy for container instances, see [To check for the `ecsInstanceRole` in the IAM console \(p. 696\)](#).

Creating a log group

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

Note

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

Using the auto-configuration feature to create a log group

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

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

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the left navigation pane, choose **Task Definitions**, **Create new Task Definition**.
3. Select your compatibility option and choose **Next Step**.
4. Choose **Add container**.
5. In the **Storage and Logging** section, for **Log configuration**, choose **Auto-configure CloudWatch Logs**.
6. Enter your `awslogs` log driver options. For more information, see [Specifying a log configuration in your task definition \(p. 286\)](#).
7. Complete the rest of the task definition wizard.

Available `awslogs` log driver options

The `awslogs` log driver supports the following options in Amazon ECS task definitions. For more information, see [CloudWatch Logs logging driver](#).

`awslogs-create-group`

Required: No

Specify whether you want the log group automatically created. If this option is not specified, it defaults to `false`.

Note

Your IAM policy must include the `logs:CreateLogGroup` permission before you attempt to use `awslogs-create-group`.

`awslogs-region`

Required: Yes

Specify the region to which the `awslogs` log driver should send your Docker logs. You can choose to send all of your logs from clusters in different regions to a single region in CloudWatch Logs so that

they are all visible in one location, or you can separate them by region for more granularity. Be sure that the specified log group exists in the region that you specify with this option.

awslogs-group

Required: Yes

You must specify a log group to which the awslogs log driver sends its log streams. For more information, see [Creating a log group \(p. 284\)](#).

awslogs-stream-prefix

Required: Optional for the EC2 launch type, required for the Fargate launch type.

The awslogs-stream-prefix option allows you to associate a log stream with the specified prefix, the container name, and the ID of the Amazon ECS task to which the container belongs. If you specify a prefix with this option, then the log stream takes the following format:

`prefix-name/container-name/ecs-task-id`

If you don't specify a prefix with this option, then the log stream is named after the container ID that is assigned by the Docker daemon on the container instance. Because it is difficult to trace logs back to the container that sent them with just the Docker container ID (which is only available on the container instance), we recommend that you specify a prefix with this option.

For Amazon ECS services, you could use the service name as the prefix, which would allow you to trace log streams to the service that the container belongs to, the name of the container that sent them, and the ID of the task to which the container belongs.

You must specify a stream-prefix for your logs in order to have your logs appear in the Log pane when using the Amazon ECS console.

awslogs-datetime-format

Required: No

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

One example of a use case for using this format is for parsing output such as a stack dump, which might otherwise be logged in multiple entries. The correct pattern allows it to be captured in a single entry.

For more information, see [awslogs-datetime-format](#).

This option always takes precedence if both `awslogs-datetime-format` and `awslogs-multiline-pattern` are configured.

Note

Multiline logging performs regular expression parsing and matching of all log messages, which may have a negative impact on logging performance.

awslogs-multiline-pattern

Required: No

This option defines a multiline start pattern using a regular expression. A log message consists of a line that matches the pattern and any following lines that don't match the pattern. Thus the matched line is the delimiter between log messages.

For more information, see [awslogs-multiline-pattern](#).

This option is ignored if `awslogs-datetime-format` is also configured.

Note

Multiline logging performs regular expression parsing and matching of all log messages. This may have a negative impact on logging performance.

`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 used for intermediate message storage.

Specifying a log configuration in your task definition

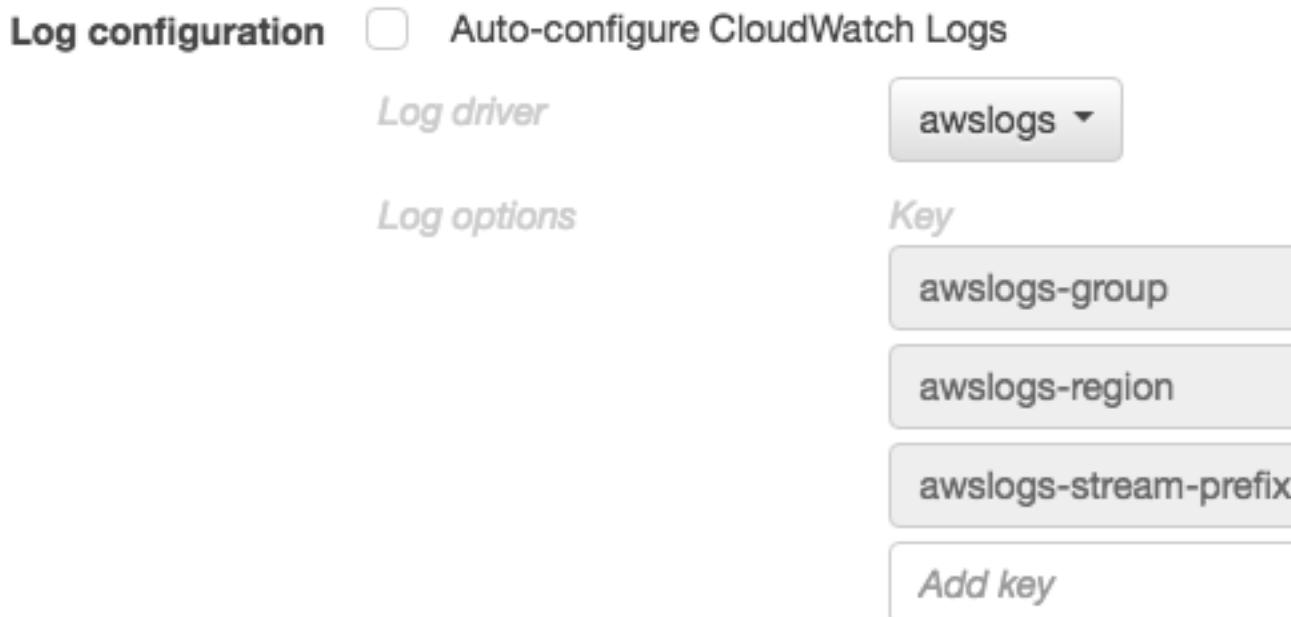
Before your containers can send logs to CloudWatch, you must specify the awslogs log driver for containers in your task definition. This section describes the log configuration for a container to use the awslogs log driver. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

The task definition JSON shown below has a `logConfiguration` object specified for each container; one for the WordPress container that sends logs to a log group called `awslogs-wordpress`, and one for a MySQL container that sends logs to a log group called `awslogs-mysql`. Both containers use the `awslogs-example` log stream prefix.

```
{  
    "containerDefinitions": [  
        {  
            "name": "wordpress",  
            "links": [  
                "mysql"  
            ],  
            "image": "wordpress",  
            "essential": true,  
            "portMappings": [  
                {  
                    "containerPort": 80,  
                    "hostPort": 80  
                }  
            ],  
            "logConfiguration": {  
                "logDriver": "awslogs",  
                "options": {  
                    "awslogs-group": "awslogs-wordpress",  
                    "awslogs-region": "us-west-2",  
                    "awslogs-stream-prefix": "awslogs-example"  
                }  
            },  
            "memory": 500,  
            "cpu": 10  
        },  
        {  
    ]}
```

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

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



After you have registered a task definition with the `awslogs` log driver in a container definition log configuration, you can run a task or create a service with that task definition to start sending logs to CloudWatch Logs. For more information, see [Run a standalone task \(p. 510\)](#) and [Creating an Amazon ECS service \(p. 546\)](#).

Viewing awslogs container logs in CloudWatch Logs

For tasks using the EC2 launch type, after your container instance role has the proper permissions to send logs to CloudWatch Logs, your container agents are updated to at least version 1.9.0, and you

have configured and started a task with containers that use the awslogs log driver, your configured containers should be sending their log data to CloudWatch Logs. You can view and search these logs in the console.

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

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, select the cluster that contains the task to view.
3. On the **Cluster: *cluster_name*** page, choose **Tasks** and select the task to view.
4. On the **Task: *task_id*** page, expand the container view by choosing the arrow to the left of the container name.
5. In the **Log Configuration** section, choose **View logs in CloudWatch**, which opens the associated log stream in the CloudWatch console.

Log Configuration	
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 \(p. 284\)](#).

The screenshot shows the AWS CloudWatch Logs console interface. At the top, there are two buttons: "Create Metric Filter" (blue) and "Actions" (dropdown). Below these is a search bar labeled "Filter: Log Group Name Prefix" with a clear button "x". Underneath is a table titled "Log Groups". It has two columns: a checkbox column and a "Log Groups" column. Two entries are listed:

- awslogs-mysql
- 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 selected time.
▶	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 de-
▶	12:56:49	AH00558: apache2: Could not reliably de-
▶	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/2016:20:06:57]

Custom log routing

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

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

Considerations

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

- FireLens for Amazon ECS is supported for tasks hosted on both AWS Fargate on Linux and Amazon EC2. Windows containers on AWS Fargate do not 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 should not allow ingress traffic on port 24224 in the security group your task uses. For tasks using 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 associated with the Amazon EC2 instance hosting

the task. For tasks using 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 \(p. 236\)](#).

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 does not support FireLens.

Required IAM permissions

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

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

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

Your task may also require the Amazon ECS task execution role under the following conditions. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

- 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 is hosted in Amazon S3, your task execution IAM role must include the `s3:GetObject` permission for the configuration file and the `s3:GetBucketLocation` permission on the Amazon S3 bucket that the file is in. For more information, see [Specifying Permissions in a Policy](#) in the *Amazon Simple Storage Service User Guide*.

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

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ]  
        }  
    ]  
}
```

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

Using 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 assigned to the FireLens container.

`FLUENT_PORT`

The port that the Fluent Forward protocol is listening on.

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

- the section called “Using the AWS for Fluent Bit image” (p. 291)
- the section called “Creating a task definition that uses a FireLens configuration” (p. 292)
- the section called “Filtering logs using regular expressions” (p. 295)
- the section called “Example task definitions” (p. 296)

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 Regions for high availability, and on Docker Hub.

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 as it is a public repository and available to be used from all AWS Regions. For more details, 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
```

Dockerhub

The AWS for Fluent Bit image is available on Docker Hub. For more details, see [AWS for Fluent Bit on Docker Hub](#).

Creating a task definition that uses a FireLens configuration

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

- A log router container containing a FireLens configuration. 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 ARN containing the permissions needed for the task to route the logs.

When creating a new task definition using the AWS Management Console, there is a FireLens integration section that makes it easy to add a log router container. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

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, so to ensure that the FireLens log router isn't reachable outside of the task you should not allow ingress traffic on port 24224 in the security group your task uses. For tasks using 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 associated with the Amazon EC2 instance hosting the task. For tasks using 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.

```
{
  "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
    }
  ]
}
```

```
        ]
    }
```

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

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

Note

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

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

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

```
{
  "containerDefinitions": [
    {
      "essential": true,
      "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit: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 you're using. For more information, see [Fluentd Config File Syntax](#) and [Fluent Bit Configuration Schema](#).

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

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

config-file-type

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

Note

Tasks 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 is mounted in the container.

Important

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

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

Important

To specify a custom configuration file that is hosted in Amazon S3, ensure you have created a task execution IAM role with the proper permissions. For more information, see [Required IAM permissions \(p. 290\)](#).

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].*$",
                "include-pattern": ".*[aeiou]$"
            }
        }
    }
]
}

```

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 \(p. 296\)](#)
- [Forwarding logs to an Amazon Kinesis Data Firehose delivery stream \(p. 297\)](#)
- [Forwarding logs to an Amazon OpenSearch Service domain \(p. 298\)](#)
- [Parsing container logs that are serialized JSON \(p. 299\)](#)
- [Forwarding to an external Fluentd or Fluent Bit \(p. 300\)](#)

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 Region it exists in. To have Fluent Bit create the log group on your behalf, specify "auto_create_group": "true". 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:stable",
            "name": "log_router",
            "firelensConfiguration": {
                "type": "fluentbit"
            },
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "firelens-container",
                    "awslogs-region": "us-west-2",
                    "awslogs-stream-prefix": "firelens"
                }
            }
        }
    ]
}

```

```

        "awslogs-region": "us-west-2",
        "awslogs-create-group": "true",
        "awslogs-stream-prefix": "firelens"
    }
},
"memoryReservation": 50
},
{
    "essential": true,
    "image": "nginx",
    "name": "app",
    "logConfiguration": {
        "logDriver": "awsfirelens",
        "options": {
            "Name": "cloudwatch",
            "region": "us-west-2",
            "log_key": "log",
            "log_group_name": "/aws/ecs/containerinsights/
$(ecs_cluster)/application",
            "auto_create_group": "true",
            "log_stream_prefix": "log_stream_name"
        }
    },
    "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-ke7thhz007jawrhzmz6mb7ite7y.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 is a JSON parser 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 will parse 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": "e54cccfcac2b8741f71877907f67879068420042828067ae0867e60a63529d35",
    "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"
}

```

With the JSON parsing, the log will look like the following:

```
{
    "source": "stdout",
    "container_id": "e54cccfcac2b8741f71877907f67879068420042828067ae0867e60a63529d35",
    "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 on 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 container definition. This allows your tasks to use images from private repositories. This feature is supported by tasks using both the Fargate or EC2 launch types.

Important

If your task definition references an image stored in Amazon ECR, this topic does not apply. For more information, see [Using Amazon ECR Images with Amazon ECS](#) in the *Amazon Elastic Container Registry User Guide*.

For tasks using the EC2 launch type, this feature requires version 1.19.0 or later of the container agent; however, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

For tasks using the Fargate launch type, this feature requires platform version 1.2.0 or later. For information, see [AWS Fargate platform versions \(p. 169\)](#).

Within your container definition, specify `repositoryCredentials` with the full ARN of the secret that you created. The secret you reference can be from a different Region than the task using it, but must be from within the same account.

Note

When using the Amazon ECS API, AWS CLI, or AWS SDK, if the secret exists in the same Region as the task you are launching then you can use either the full ARN or name of the secret. When using the AWS Management Console, the full ARN of the secret must be specified.

The following is a snippet of a task definition showing the required parameters:

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

Note

Another method of enabling private registry authentication uses Amazon ECS container agent environment variables to authenticate to private registries. This method is only supported for tasks using the EC2 launch type. For more information, see [Private registry authentication for container instances \(p. 468\)](#).

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 \(p. 691\)](#).

To provide access to the secrets that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see [Adding and Removing IAM Policies](#).

- `secretsmanager:GetSecretValue`
- `kms:Decrypt`—Required only if your key uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

An example inline policy adding the permissions is shown below.

```
{
    "Version": "2012-10-17",
```

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

Enabling private registry authentication

To create a basic secret

Use AWS Secrets Manager to create a secret for your private registry credentials.

1. Open the AWS Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.
3. For **Select secret type**, choose **Other type of secrets**.
4. Select **Plaintext** and enter your private registry credentials using the following format:

```
{
    "username" : "privateRegistryUsername",
    "password" : "privateRegistryPassword"
}
```

5. Choose **Next**.
6. For **Secret name**, type an optional path and name, such as **production/MyAwesomeAppSecret** or **development/TestSecret**, and choose **Next**. You can optionally add a description to help you remember the purpose of this secret later.

The secret name must be ASCII letters, digits, or any of the following characters: /_+=.=@-

7. (Optional) At this point, you can configure rotation for your secret. For this procedure, leave it at **Disable automatic rotation** and choose **Next**.

For information about how to configure rotation on new or existing secrets, see [Rotating Your AWS Secrets Manager Secrets](#).

8. Review your settings, and then choose **Store secret** to save everything you entered as a new secret in Secrets Manager.

To create a task definition that uses private registry authentication

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Task Definitions**.
3. On the **task definitions** page, choose **Create new task definition**.
4. On the **Select launch type compatibility** page, choose the launch type for your tasks and then **Next step**.

Note

This step only applies to regions that currently support Amazon ECS using AWS Fargate. For more information, see [Amazon ECS on AWS Fargate \(p. 161\)](#).

5. For **task definition Name**, type a name for your task definition. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
6. For **Task execution role**, either select your existing task execution role or choose **Create new role** to have one created for you. This role authorizes Amazon ECS to pull private images for your task. For more information, see [Required IAM permissions for private registry authentication \(p. 301\)](#).

Important

If the **Task execution role** field does not appear, choose **Configure via JSON** and manually add the `executionRoleArn` field to specify your task execution role. The following shows the syntax:

```
"executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"
```

7. For each container to create in your task definition, complete the following steps:
 - a. In the **Container Definitions** section, choose **Add container**.
 - b. For **Container name**, type a name for your container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - c. For **Image**, type the image name or path to your private image. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - d. Select the **Private repository authentication** option.
 - e. For **Secrets manager ARN**, enter the full Amazon Resource Name (ARN) of the secret that you created earlier. The value must be between 20 and 2048 characters.
 - f. Fill out the remaining required fields and any optional fields to use in your container definitions. More container definition parameters are available in the **Advanced container configuration** menu. For more information, see [Task definition parameters \(p. 209\)](#).
 - g. Choose **Add**.
8. When your containers are added, choose **Create**.

Specifying sensitive data

Amazon ECS enables you to inject sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition.

Secrets can be exposed to a container in the following ways:

- To inject sensitive data into your containers as environment variables, use the `secrets` container definition parameter.
- To reference sensitive information in the log configuration of a container, use the `secretOptions` container definition parameter.

Topics

- [Specifying sensitive data using Secrets Manager \(p. 303\)](#)
- [Specifying sensitive data using Systems Manager Parameter Store \(p. 310\)](#)

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

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

- For Amazon ECS tasks on AWS Fargate, the following should be considered:
 - To inject the full content of a secret as an environment variable or in a log configuration, you must use platform version 1.3.0 or later. For information, see [AWS Fargate platform versions \(p. 169\)](#).
 - 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 \(p. 169\)](#).
- For Amazon ECS tasks on EC2, the following should be considered:
 - To inject a secret using a specific JSON key or version of a secret, your container instance must have version 1.37.0 or later of the container agent. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

To inject the full contents of a secret as an environment variable or to inject a secret in a log configuration, your container instance must have version 1.22.0 or later of the container agent.

- Only secrets that store text data, which are secrets created with the `SecretString` parameter of the [CreateSecret](#) API, are supported. Secrets that store binary data, which are secrets created with the `SecretBinary` parameter of the [CreateSecret](#) API are not supported.
- When using a task definition that references Secrets Manager secrets to retrieve sensitive data for your containers, if you are also using interface VPC endpoints, you must create the interface VPC endpoints for Secrets Manager. For more information, see [Using Secrets Manager with VPC Endpoints](#) in the *AWS Secrets Manager User Guide*.
- Sensitive data is injected into your container when the container is initially started. If the secret is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- The VPC your task uses must have DNS resolution enabled.
- For Windows tasks that are configured to use the `awslogs` logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. This can be done with User Data using the following syntax:

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

Required IAM permissions for Amazon ECS secrets

To use this feature, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary Secrets Manager resources. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

To provide access to the Secrets Manager secrets that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see [Adding and Removing IAM Policies](#).

- `secretsmanager:GetSecretValue`—Required if you are referencing a Secrets Manager secret.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions.

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

Injecting sensitive data as an environment variable

Within your container definition, you can specify the following:

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

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

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

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

`json-key`

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

`version-stage`

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

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

version-id

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

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

For a full tutorial on creating a Secrets Manager secret and injecting it into a container as an environment variable, see [Tutorial: Specifying sensitive data using Secrets Manager secrets \(p. 764\)](#).

Example container definitions

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

Example referencing a full secret

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

```
{  
    "containerDefinitions": [  
        {  
            "secrets": [  
                {  
                    "name": "environment\_variable\_name",  
                    "valueFrom": "arn:aws:secretsmanager:region:aws\_account\_id:secret:secret\_name-AbCdEf"  
                }  
            ]  
        }  
    ]  
}
```

Example referencing a specific key within a secret

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

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

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

```
{  
    "containerDefinitions": [  
        {  
            "secrets": [  
                {  
                    "name": "appauthexample",  
                    "valueFrom": "arn:aws:secretsmanager:region:aws\_account\_id:secret:appauthexample-AbCdEf:username1"  
                }  
            ]  
        }  
    ]  
}
```

```

        "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::AWSNEXT:"  
    }]  
  }
}
```

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

```
{
  "containerDefinitions": [{  
    "secrets": [{  
      "name": "environment_variable_name",  
      "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-  
AbCdEf:::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"  
    }]  
  }
}
```

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

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

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

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

```
{
  "containerDefinitions": [
    "secrets": [
      {
        "name": "environment_variable_name",
        "valueFrom": "arn:aws:secretsmanager:region:aws_account_id:secret:appauthexample-
AbCdEf:username1::9d4cb84b-ad69-40c0-a0ab-cead3EXAMPLE"
      }
    ]
  }
}
```

Injecting sensitive data in a log configuration

Within your container definition, when specifying a `logConfiguration` you can specify `secretOptions` with the name of the log driver option to set in the container and the full ARN of the Secrets Manager secret containing the sensitive data to present to the container.

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

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

Creating an AWS Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. For more information, see [Creating a Basic Secret](#) in the *AWS Secrets Manager User Guide*.

To create a basic secret

Use Secrets Manager to create a secret for your sensitive data.

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.
3. For **Select secret type**, choose **Other type of secrets**.

4. Specify the details of your custom secret as **Key** and **Value** pairs. For example, you can specify a key of `UserName`, and then supply the appropriate user name as its value. Add a second key with the name of `Password` and the password text as its value. You could also add entries for a database name, server address, TCP port, and so on. You can add as many pairs as you need to store the information you require.

Alternatively, you can choose the **Plaintext** tab and enter the secret value in any way you like.

5. Choose the AWS KMS encryption key that you want to use to encrypt the protected text in the secret. If you don't choose one, Secrets Manager checks to see if there's a default key for the account, and uses it if it exists. If a default key doesn't exist, Secrets Manager creates one for you automatically. You can also choose **Add new key** to create a custom 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 you entered as a new secret in Secrets Manager.

Creating a task definition that references sensitive data

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

Note

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

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 \(p. 301\)](#).

Important

If the **Task execution role** field does not appear, choose **Configure via JSON** and manually add the `executionRoleArn` field to specify your task execution role. The following code shows the syntax:

```
"executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"
```

6. For each container to create in your task definition, complete the following steps:
 - a. Under **Container Definitions**, choose **Add container**.

- b. For **Container name**, type a name for your container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - c. For **Image**, type the image name or path to your private image. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - d. Expand **Advanced container configuration**.
 - e. For sensitive data to inject as environment variables, under **Environment**, for **Environment variables**, complete the following fields:
 - i. For **Key**, enter the name of the environment variable to set in the container. This corresponds to the `name` field in the `secrets` section of a container definition.
 - ii. For **Value**, choose **ValueFrom**. For **Add value**, enter the ARN of the Secrets Manager secret that contains the data to present to your container as an environment variable.
 - f. For sensitive data referenced in the log configuration for a container, under **Storage and Logging**, for **Log configuration**, complete the following fields:
 - i. Clear the **Auto-configure CloudWatch Logs** option.
 - ii. Under **Log options**, for **Key**, enter the name of the log configuration option to set.
 - iii. For **Value**, choose **ValueFrom**. For **Add value**, enter the full ARN of the Secrets Manager secret that contains the data to present to your log configuration as a log option.
 - g. Fill out the remaining required fields and any optional fields to use in your container definitions. More container definition parameters are available in the **Advanced container configuration** menu. For more information, see [Task definition parameters \(p. 209\)](#).
 - h. Choose **Add**.
7. When your containers are added, choose **Create**.

Specifying sensitive data using Systems Manager Parameter Store

Amazon ECS enables you to inject sensitive data into your containers by storing your sensitive data in AWS Systems Manager Parameter Store parameters and then referencing them in your container definition.

Topics

- [Considerations for specifying sensitive data using Systems Manager Parameter Store \(p. 310\)](#)
- [Required IAM permissions for Amazon ECS secrets \(p. 311\)](#)
- [Injecting sensitive data as an environment variable \(p. 312\)](#)
- [Injecting sensitive data in a log configuration \(p. 312\)](#)
- [Creating an AWS Systems Manager Parameter Store parameter \(p. 313\)](#)
- [Creating a Task Definition that References sensitive data \(p. 313\)](#)

Considerations for specifying sensitive data using Systems Manager Parameter Store

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

- For tasks that use the Fargate launch type, this feature requires that your task use platform version 1.3.0 or later (for Linux) or 1.0.0 or later (for Windows). For information, see [AWS Fargate platform versions \(p. 169\)](#).

- For tasks that use the EC2 launch type, this feature requires that your container instance have version 1.22.0 or later of the container agent. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).
- Sensitive data is injected into your container when the container is initially started. If the secret or Parameter Store parameter is subsequently updated or rotated, the container will not receive the updated value automatically. You must either launch a new task or if your task is part of a service you can update the service and use the **Force new deployment** option to force the service to launch a fresh task.
- For Windows tasks that are configured to use the awslogs logging driver, you must also set the `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE` environment variable on your container instance. This can be done with User Data using the following syntax:

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

Required IAM permissions for Amazon ECS secrets

To use this feature, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary AWS Systems Manager resources. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

Important

For tasks that use the EC2 launch type, you must use the ECS agent configuration variable `ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE=true` to use this feature. You can add it to the `./etc/ecs/ecs.config` file during container instance creation or you can add it to an existing instance and then restart the ECS agent. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

To provide access to the AWS Systems Manager Parameter Store parameters that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see [Adding and Removing IAM Policies](#).

- `ssm:GetParameters`—Required if you are referencing a Systems Manager Parameter Store parameter in a task definition.
- `secretsmanager:GetSecretValue`—Required if you are referencing a Secrets Manager secret either directly or if your Systems Manager Parameter Store parameter is referencing a Secrets Manager secret in a task definition.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions:

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

```

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

```

Injecting sensitive data as an environment variable

Within your container definition, specify secrets with the name of the environment variable to set in the container and the full ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

The following is a snippet of a task definition showing the format when referencing 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 ARN of the Systems Manager Parameter Store parameter containing the sensitive data to present to the container.

Important

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

The following is a snippet of a task definition showing the format when referencing 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**, **Create parameter**.
3. For **Name**, type a hierarchy and a parameter name. For example, type /test/database_password.
4. For **Description**, type an optional description.
5. For **Type**, choose **String**, **StringList**, or **SecureString**.

Note

- If you choose **SecureString**, the **KMS key ID** field appears. If you don't provide a KMS key ID, a KMS key ARN, an alias name, or an alias ARN, then the system uses alias/aws/ssm, which 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 that References sensitive data

You can use the Amazon ECS console to create a task definition that references a Systems Manager Parameter Store parameter.

To create a task definition that specifies a secret

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **task definitions**, **Create New Task Definition**.
3. On the **Select launch type compatibility** page, choose the launch type for your tasks and choose **Next step**.

Note

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

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 \(p. 301\)](#).

Important

If the **Task execution role** field does not appear, choose **Configure via JSON** and manually add the `executionRoleArn` field to specify your task execution role. The following code shows the syntax:

```
"executionRoleArn": "arn:aws:iam::aws_account_id:role/ecsTaskExecutionRole"
```

6. For each container to create in your task definition, complete the following steps:
 - a. Under **Container Definitions**, choose **Add container**.
 - b. For **Container name**, type a name for your container. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - c. For **Image**, type the image name or path to your private image. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
 - d. Expand **Advanced container configuration**.
 - e. For sensitive data to inject as environment variables, under **Environment**, for **Environment variables**, complete the following fields:
 - i. For **Key**, enter the name of the environment variable to set in the container. This corresponds to the `name` field in the `secrets` section of a container definition.
 - ii. For **Value**, choose **ValueFrom**. For **Add value**, enter the full ARN of the AWS Systems Manager Parameter Store parameter that contains the data to present to your container as an environment variable.
 - 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.
 - 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 \(p. 209\)](#).
 - h. Choose **Add**.
7. When your containers are added, choose **Create**.

Specifying environment variables

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

- Individually using the `environment` container definition parameter. This maps to the `--env` option to [docker run](#).
- In bulk, using the `environmentFiles` container definition parameter to list one or more files containing the environment variables. The file must be hosted in Amazon S3. This maps to the `--env-file` option to [docker run](#).

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

We do not 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

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

- For Amazon ECS tasks on Amazon EC2, your container instances require version 1.39.0 or later of the container agent to use this feature. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).
- For Amazon ECS tasks on AWS Fargate, your tasks must use platform version 1.4.0 or later (Linux) or 1.0.0 or later (Windows) to use this feature. For more information, see [AWS Fargate platform versions \(p. 169\)](#).
- The file must use the .env file extension and UTF-8 encoding.
- Each line in an environment file should contain an environment variable in VARIABLE=VALUE format. Spaces or quotation marks are included as part of the values. Lines beginning with # are treated as comments and are ignored. For more information on the environment variable file syntax, see [Declare default environment variables in file](#).

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

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

- If there are environment variables specified using the environment parameter in a container definition, they take precedence over the variables contained within an environment file.
- If multiple environment files are specified and they contain the same variable, they are 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 is used, and any other environment files specified in a 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 \(p. 691\)](#).

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

- s3:GetObject
- s3:GetBucketLocation

An example inline policy adding the permissions is shown.

```
{  
    "Version": "2012-10-17",
```

```

"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::examplebucket/folder_name/env_file_name"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation"
        ],
        "Resource": [
            "arn:aws:s3:::examplebucket"
        ]
    }
]
}

```

Example task definitions

This section provides some task definition examples that you can use to start creating your own task definitions. For more information, see [Task definition parameters \(p. 209\)](#) and [Creating a task definition using the new console \(p. 196\)](#).

For additional task definition examples, see [AWS Sample Task Definitions](#) on GitHub.

Topics

- [Example: Webserver \(p. 317\)](#)
- [Example: splunk log driver \(p. 319\)](#)
- [Example: fluentd log driver \(p. 319\)](#)
- [Example: gelf log driver \(p. 320\)](#)
- [Example: Amazon ECR image and task definition IAM role \(p. 320\)](#)
- [Example: Entrypoint with command \(p. 321\)](#)
- [Example: Container dependency \(p. 321\)](#)

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",
"platformFamily": "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 \(p. 303\)](#).

```
"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 \(p. 303\)](#).

```
"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 \(p. 227\)](#).

```
"containerDefinitions": [
    "logConfiguration": {
        "logDriver": "gelf",
        "options": {
            "gelf-address": "udp://logstash-service-address:5000",
            "tag": "gelf task demo"
        }
    },
    "entryPoint": [],
    "portMappings": [
        {
            "hostPort": 5000,
            "protocol": "udp",
            "containerPort": 5000
        },
        {
            "hostPort": 5000,
            "protocol": "tcp",
            "containerPort": 5000
        }
    ]
],
```

Example: Amazon ECR image and task definition IAM role

The following example uses an Amazon ECR image called `aws-nodejs-sample` with the `v1` tag from the `123456789012.dkr.ecr.us-west-2.amazonaws.com` registry. The container in this task inherits IAM permissions from the `arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole` role. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

```
{
    "containerDefinitions": [
        {
            "name": "sample-app",
            "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/aws-nodejs-sample:v1",
            "memory": 200,
            "cpu": 10,
            "essential": true
        }
    ],
    "family": "example_task_3",
    "taskRoleArn": "arn:aws:iam::123456789012:role/AmazonECSTaskS3BucketRole"
}
```

Example: Entrypoint with command

The following example demonstrates the syntax for a Docker container that uses an entry point and a command argument. This container pings google.com four times and then exits.

```
{
  "containerDefinitions": [
    {
      "memory": 32,
      "essential": true,
      "entryPoint": [
        "ping"
      ],
      "name": "alpine_ping",
      "readonlyRootFilesystem": true,
      "image": "alpine:3.4",
      "command": [
        "-c",
        "4",
        "google.com"
      ],
      "cpu": 16
    }
  ],
  "family": "example_task_2"
}
```

Example: Container dependency

This example demonstrates the syntax for a task definition with multiple containers where container dependency is specified. In the following task definition, the `envoy` container must reach a healthy status, determined by the required container healthcheck parameters, before the `app` container will start. For more information, see [Container dependency \(p. 236\)](#).

```
{
  "family": "appmesh-gateway",
  "platformFamily": "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"
}

```

Updating a task definition

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

To create a task definition revision

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, choose the Region that contains your task definition.
3. In the navigation pane, choose **task definitions**.

4. On the **task definitions** page, select the box to the left of the task definition to revise and choose **Create new revision**.
5. On the **Create new revision of task definition** page, make changes. For example, to change the existing container definitions (such as the container image, memory limits, or port mappings), select the container, make the changes, and then choose **Update**.
6. Verify the information and choose **Create**.
7. If your task definition is used in a service, update your service with the updated task definition. For more information, see [Updating a service \(p. 559\)](#).

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 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.

Use the following procedure to deregister a task definition revision.

To deregister a task definition revision

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, choose the region that contains your task definition.
3. In the navigation pane, choose **task definitions**.
4. On the **task definitions** page, choose the task definition family that contains one or more revisions that you want to deregister.
5. On the **task definition Name** page, select the box to the left of each task definition revision you want to deregister.
6. Choose **Actions, Deregister**.
7. Verify the information in the **Deregister task definition** window, and choose **Deregister** to finish.

Account settings

Amazon ECS provides 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 \(p. 325\)](#).

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.

AWSVPC trunking

Resource name: `awsvpcTrunking`

Amazon ECS supports launching container instances with increased elastic network interface (ENI) density using supported Amazon EC2 instance types. When you use these instance types and opt in to the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks using the `awsvpc` network mode on each container instance. Using this feature, a `c5.large` instance with `awsvpcTrunking` enabled has an increased ENI limit of ten. The container instance has a primary network interface, and Amazon ECS creates and attaches a "trunk" network interface to the container instance. The primary

network interface and the trunk network interface don't count against the ENI limit. Therefore, this configuration allows you to launch ten tasks on the container instance instead of the current two tasks. For more information, see [Elastic network interface trunking \(p. 372\)](#).

Only resources launched after opting in receive the increased ENI limits. All existing resources are not affected. To transition a container instance to the increased ENI limits, the container instance must be drained and a new container instance registered to the cluster.

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 \(p. 645\)](#).

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.

For clusters containing tasks or services using the EC2 launch type, your container instances must be running version 1.29.0 or later of the Amazon ECS agent to use Container Insights. For more information, see [Amazon ECS container agent versions \(p. 440\)](#).

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 \(p. 282\)](#).

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 \(p. 328\)](#).

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 \(p. 325\)](#)
- [ARN and resource ID format timeline \(p. 326\)](#)
- [Viewing account settings \(p. 327\)](#)
- [Modifying account settings \(p. 328\)](#)

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 \(p. 328\)](#).

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

Resource type	ARN
Container instance	Old: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:container-instance/<i>container-instance-id</i></code> New: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:container-instance/<i>cluster-name</i>/<i>container-instance-id</i></code>
Amazon ECS service	Old: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:service/<i>service-name</i></code> New: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:service/<i>cluster-name</i>/<i>service-name</i></code>
Amazon ECS task	Old: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:task/<i>task-id</i></code> New: <code>arn:aws:ecs:<i>region</i>:<i>aws_account_id</i>:task/<i>cluster-name</i>/<i>task-id</i></code>

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 \(p. 328\)](#).

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**, **AWSVPC Trunking**, and **CloudWatch Container Insights** sections, you can view your opt-in status for each account setting for the authenticated IAM user and role.

To view your account settings (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::aws_account_id:user/principalName --effective-settings --region us-east-1
```

- [Get-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

```
Get-ECSAccountSetting -PrincipalArn arn:aws:iam::aws_account_id:user/principalName -
EffectiveSetting true -Region us-east-1
```

Modifying account settings

You can use the AWS Management Console and AWS CLI tools to modify 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**, **AWSVPC Trunking**, and **CloudWatch Container Insights** sections, you can select or deselect the check boxes for each account setting for the authenticated IAM user and role. Choose **Save** once finished.

Important

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

5. On the confirmation screen, choose **Confirm** to save the selection.

To modify the default account settings for all IAM users or roles on your account (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
```

Amazon ECS container instances

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

Note

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

Amazon ECS supports the following container instance types.

- Linux
- Windows
- External, such as an on-premises VM

Topics

- [Container instance concepts \(p. 330\)](#)
- [Container instance lifecycle \(p. 331\)](#)
- [Check the instance IAM role for your account \(p. 332\)](#)
- [Linux instances \(p. 332\)](#)
- [Windows instances \(p. 387\)](#)
- [External instances \(Amazon ECS Anywhere\) \(p. 414\)](#)
- [Monitoring your container instances \(p. 426\)](#)
- [Container instance draining \(p. 428\)](#)
- [Deregister an Amazon EC2 backed container instance \(p. 429\)](#)

Container instance concepts

- Your container instance must be running the Amazon ECS container agent. The container agent is able to register the instance into one of your clusters. If you are using an Amazon ECS-optimized AMI, the agent is already installed. To use a different operating system, install the agent. For more information, see [Amazon ECS container agent \(p. 431\)](#).
- Because the Amazon ECS container agent makes calls to Amazon ECS on your behalf, you must launch container instances with an IAM role that authenticates to your account and provides the required resource permissions. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
- Beginning with Linux Amazon ECS-optimized AMI version 20200430 and later, the Amazon EC2 Instance Metadata Service Version 2 (IMDSv2) is supported on your container instances. For Amazon ECS-optimized AMIs prior to version 20200430, Amazon EC2 Instance Metadata Service Version 1 (IMDSv1) is supported. For more information, see [Configuring the instance metadata service](#) in the [Amazon EC2 User Guide for Linux Instances](#).
- If any of the containers associated with your tasks require external connectivity, you can map their network ports to ports on the host Amazon ECS container instance so they are reachable from the internet. Your container instance security group must allow inbound access to the ports you want to expose. For more information, see [Create a Security Group](#) in the [Amazon VPC Getting Started Guide](#).
- We strongly recommend launching your container instances inside a VPC, because Amazon VPC delivers more control over your network and offers more extensive configuration capabilities. For more information, see [Amazon EC2 and Amazon Virtual Private Cloud](#) in the [Amazon EC2 User Guide for Linux Instances](#).

- Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 714\)](#).

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 504\)](#) in this guide. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters \(p. 727\)](#).

- The type of Amazon EC2 instance that you choose for your container instances determines the resources available in your cluster. Amazon EC2 provides different instance types, each with different CPU, memory, storage, and networking capacity that you can use to run your tasks. For more information, see [Amazon EC2 Instances](#).
- Because each container instance has unique state information that is stored locally on the container instance and within Amazon ECS:
 - You should not deregister an instance from one cluster and re-register it into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances with the latest Amazon ECS-optimized Amazon Linux 2 AMI in the new cluster. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances* and [Launching an Amazon ECS Linux container instance \(p. 364\)](#).
 - You cannot stop a container instance and change its instance type. Instead, we recommend that you terminate the container instance and launch a new container instance with the desired instance size and the latest Amazon ECS-optimized Amazon Linux 2 AMI in your desired cluster. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances* and [Launching an Amazon ECS Linux container instance \(p. 364\)](#) in this guide.

Container instance lifecycle

When the Amazon ECS container agent registers an Amazon EC2 instance into your cluster, the Amazon EC2 instance reports its status as `ACTIVE` and its agent connection status as `TRUE`. This container instance can accept `RunTask` requests.

If you stop (not terminate) an Amazon ECS container instance, the status remains `ACTIVE`, but the agent connection status transitions to `FALSE` within a few minutes. Any tasks that were running on the container instance stop. If you start the container instance again, the container agent reconnects with the Amazon ECS service, and you are able to run tasks on the instance again.

Important

If you stop and start a container instance, or reboot that instance, some older versions of the Amazon ECS container agent register the instance again without deregistering the original container instance ID. In this case, Amazon ECS lists more container instances in your cluster than you actually have. (If you have duplicate container instance IDs for the same Amazon EC2 instance ID, you can safely deregister the duplicates that are listed as `ACTIVE` with an agent connection status of `FALSE`.) This issue is fixed in the current version of the Amazon ECS container agent. For more information about updating to the current version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

If you change the status of a container instance to `DRAINING`, new tasks are not placed on the container instance. Any service tasks running on the container instance are removed, if possible, so that you can perform system updates. For more information, see [Container instance draining \(p. 428\)](#).

If you deregister or terminate a container instance, the container instance status changes to `INACTIVE` immediately, and the container instance is no longer reported when you list your container instances.

However, you can still describe the container instance for one hour following termination. After one hour, the instance description is no longer available.

Check the instance IAM role for your account

The Amazon ECS container agent makes calls to the Amazon ECS APIs on your behalf. Container instances that run the agent require an IAM policy and role for the service to know that the agent belongs to you.

In most cases, the Amazon ECS instance role is automatically created for you in the console first-run experience. You can use the following procedure to check and see if your account already has an Amazon ECS service role.

To check for the `ecsInstanceRole` in the IAM console

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsInstanceRole`. If the role exists, you do not need to create it. If the role does not exist, follow the procedures in [Amazon ECS container instance IAM role \(p. 695\)](#) to create the role.

Linux instances

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

Note

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

The following Linux container instance operating systems are available:

- **Amazon Linux:** This is a general purpose operating system.
- **Bottlerocket:** This is an operating system that is optimized for container workloads and that has a focus on security. It does not include a package manager and is immutable by default. For information about the security features and guidance, see [Security Features](#) and [Security Guidance](#) on the GitHub website.

Topics

- [Amazon ECS-optimized AMI \(p. 333\)](#)
- [Using Bottlerocket with Amazon ECS \(p. 362\)](#)
- [Launching an Amazon ECS Linux container instance \(p. 364\)](#)
- [Bootstrapping container instances with Amazon EC2 user data \(p. 368\)](#)
- [Starting a task at container instance launch time \(p. 369\)](#)
- [Elastic network interface trunking \(p. 372\)](#)
- [Container Instance Memory Management \(p. 382\)](#)
- [Connect to your container instance \(p. 384\)](#)
- [Manage container instances remotely using AWS Systems Manager \(p. 385\)](#)

Amazon ECS-optimized AMI

An Amazon ECS container instance specification consists of the following components.

Required

- A modern Linux distribution running at least version 3.10 of the Linux kernel.
- The Amazon ECS container agent (preferably the latest version). For more information, see [Amazon ECS container agent \(p. 431\)](#).
- A Docker daemon running at least version 1.9.0, and any Docker runtime dependencies. For more information, see [Check runtime dependencies](#) in the Docker documentation.

Note

For the best experience, we recommend the Docker version that ships with and is tested with the corresponding Amazon ECS container agent version that you are using.

Recommended

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

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

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

The Amazon ECS-optimized AMI metadata, including the AMI name, Amazon ECS container agent version, and ECS runtime version which includes the Docker version, for each variant can be retrieved programmatically. For more information, see [Retrieving Amazon ECS-Optimized AMI metadata \(p. 339\)](#).

The following variants of the Amazon ECS-optimized AMI are available for your Amazon EC2 instances.

- **Amazon ECS-optimized Amazon Linux 2 AMI** – Recommended for launching your Amazon ECS container instances in most cases.

The Amazon ECS-optimized Amazon Linux 2 AMI does not come with the AWS CLI preinstalled.
- **Amazon ECS-optimized Amazon Linux 2 (arm64) AMI** – Based on Amazon Linux 2, this AMI is recommended for use when launching your instances, which are powered by Arm-based AWS Graviton/Graviton 2 Processors. For more information, see [General Purpose Instances](#) in the [Amazon EC2 User Guide for Linux Instances](#).

The Amazon ECS-optimized Amazon Linux 2 (arm64) AMI does not come with the AWS CLI preinstalled.

- **Amazon ECS GPU-optimized AMI** – Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 GPU-based instances. It comes pre-configured with NVIDIA kernel drivers and a Docker GPU runtime which makes running workloads that take advantage of GPUs on Amazon ECS. For more information, see [Working with GPUs on Amazon ECS \(p. 250\)](#).

The Amazon ECS GPU-optimized AMI does not come with the AWS CLI preinstalled.

- **Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI** – Based on Amazon Linux 2, this AMI is recommended for use when launching your Amazon EC2 Inf1 instances. It comes pre-configured

with AWS Inferentia drivers and the AWS Neuron runtime for Docker which makes running machine learning inference workloads easier on Amazon ECS. For more information, see [Working with inference workloads on Amazon ECS \(p. 253\)](#).

The Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI does not come with the AWS CLI preinstalled.

- **Amazon ECS-optimized Amazon Linux AMI** – This AMI is based off of Amazon Linux. We recommend that you migrate your workloads to the Amazon ECS-optimized Amazon Linux 2 AMI. Support for the Amazon ECS-optimized Amazon Linux AMI is the same as the Amazon Linux AMI. For more information, see [Amazon Linux AMI](#).

Important

On April 15, 2021, the Amazon ECS-optimized Amazon Linux AMI ended its standard support phase and entered a maintenance support phase. In the maintenance support phase, Amazon ECS will continue providing critical and important security updates for a reduced list of packages. During this period, Amazon ECS will no longer add support for new EC2 instance types, new services and features, and new packages. Instead, Amazon ECS will provide updates only for critical and important security fixes that apply to a reduced set of packages. Maintenance support period will end on June 30, 2023.

Linux Amazon ECS-optimized AMI

The following are the details for retrieving the AMI IDs for each of the Linux variants of the Amazon ECS-optimized AMI.

Amazon Linux 2

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

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

The following table provides a link to retrieve the current Amazon ECS-optimized Amazon Linux 2 AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Africa (Cape Town)	af-south-1	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Osaka)	ap-northeast-3	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID

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

For a full list of current and previous versions of the Amazon ECS-optimized Amazon Linux 2 AMI and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package, see [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#).

Amazon Linux 2 (arm64)

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

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

The following table provides a link to retrieve the current Amazon ECS-optimized Amazon Linux 2 (arm64) AMI IDs by Region.

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

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

For a full list of current and previous versions of the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package, see [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#).

Amazon Linux 2 (GPU)

You can retrieve the current Amazon ECS GPU-optimized AMI using the AWS CLI with the following command:

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

The following table provides a link to retrieve the current Amazon ECS GPU-optimized AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Africa (Cape Town)	af-south-1	View AMI ID

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

For a full list of current and previous versions of the Amazon ECS GPU-optimized AMI and their corresponding versions of the Amazon ECS container agent, Docker, the `ecs-init` package, and NVIDIA driver see [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#).

Amazon Linux 2 (Inferentia)

You can retrieve the current Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI using the AWS CLI with the following command:

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

The following table provides a link to retrieve the current Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI IDs by Region.

Region name	Region	AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US East (Ohio)	us-east-2	View AMI ID

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

For a full list of current and previous versions of the Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package see [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#).

Amazon Linux AMI

Important

On April 15, 2021, the Amazon ECS-optimized Amazon Linux AMI ended its standard support phase and entered a maintenance support phase. In the maintenance support phase, Amazon ECS will continue providing critical and important security updates for a reduced list of packages. During this period, Amazon ECS will no longer add support for new EC2 instance types, new services and features, and new packages. Instead, Amazon ECS will provide updates only for critical and important security fixes that apply to a reduced set of packages. Maintenance support period will end on June 30, 2023.

You can retrieve the current Amazon ECS-optimized Amazon Linux AMI using the AWS CLI with the following command:

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

The following table provides a link to retrieve the current Amazon ECS-optimized Amazon Linux AMI IDs by Region.

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

For a full list of current and previous versions of the Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package see [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#).

Retrieving Amazon ECS-Optimized AMI metadata

The AMI ID, image name, operating system, container agent version, and runtime version for each variant of the Amazon ECS-optimized AMIs can be programmatically retrieved by querying the Systems

Manager Parameter Store API. For more information about the Systems Manager Parameter Store API, see [GetParameters](#) and [GetParametersByPath](#).

Note

Your user account must have the following IAM permissions to retrieve the Amazon ECS-optimized AMI metadata. These permissions have been added to the `AmazonECS_FullAccess` IAM policy.

- `ssm:GetParameters`
- `ssm:GetParameter`
- `ssm:GetParametersByPath`

Systems Manager Parameter Store parameter format

The following is the format of the parameter name for each Amazon ECS-optimized AMI variant.

Linux Amazon ECS-optimized AMIs

- Amazon Linux 2 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/<version>
```

- Amazon Linux 2 (arm64) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/arm64/<version>
```

- Amazon Linux 2 (GPU) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/gpu/<version>
```

- Amazon Linux 2 (Inferentia) AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/inf/<version>
```

- Amazon Linux AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux/<version>
```

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

The following parameter name format retrieves the image ID of the latest stable Amazon ECS-optimized Amazon Linux 2 AMI by using the sub-parameter `image_id`.

```
/aws/service/ecs/optimized-ami/amazon-linux-2/<recommended>/image_id
```

The following parameter name format retrieves the metadata of a specific Amazon ECS-optimized AMI version by specifying the AMI name.

- Amazon ECS-optimized Amazon Linux 2 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/<ami>
```

Note

All versions of the Amazon ECS-optimized Amazon Linux 2 AMI are available for retrieval. Only Amazon ECS-optimized AMI versions amzn-ami-2017.09.1-amazon-ecs-optimized (Linux) and later can be retrieved. For more information, see [Amazon ECS-optimized AMI versions \(p. 342\)](#).

Examples

The following examples show ways in which you can retrieve the metadata for each Amazon ECS-optimized AMI variant.

Retrieving the metadata of the latest stable Amazon ECS-optimized AMI

You can retrieve the latest stable Amazon ECS-optimized AMI using the AWS CLI with the following AWS CLI commands.

Linux Amazon ECS-optimized AMIs

- **For the Amazon ECS-optimized Amazon Linux 2 AMIs:**

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

- **For the Amazon ECS-optimized Amazon Linux 2 (arm64) AMIs:**

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/arm64/  
recommended --region us-east-1
```

- **For the Amazon ECS GPU-optimized AMIs:**

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

- **For the Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMIs:**

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

- **For the Amazon ECS-optimized Amazon Linux AMIs:**

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

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

Retrieving the metadata of a specific Amazon ECS-optimized Amazon Linux 2 AMI version

Retrieve the metadata of a specific Amazon ECS-optimized Amazon Linux AMI version using the AWS CLI with the following AWS CLI command. Replace the AMI name with the name of the Amazon ECS-optimized Amazon Linux AMI to retrieve. For more information about the available versions, see [Amazon ECS-optimized AMI versions \(p. 342\)](#).

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-ecs-  
hvm-2.0.20200928-x86_64-ebs --region us-east-1
```

Retrieving the Amazon ECS-optimized Amazon Linux 2 AMI metadata using the Systems Manager GetParametersByPath API

Retrieve the Amazon ECS-optimized Amazon Linux 2 AMI metadata with the Systems Manager GetParametersByPath API using the AWS CLI with the following command.

```
aws ssm get-parameters-by-path --path /aws/service/ecs/optimized-ami/amazon-linux-2/ --region us-east-1
```

Retrieving the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2 AMI

You can retrieve the image ID of the latest recommended Amazon ECS-optimized Amazon Linux 2 AMI ID by using the sub-parameter `image_id`.

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

To retrieve the `image_id` value only, you can query the specific parameter value; for example:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-2/recommended/image_id --region us-east-1 --query "Parameters[0].Value"
```

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

You can reference the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template by referencing the Systems Manager parameter store name.

Linux example

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

Amazon ECS-optimized AMI versions

This topic lists the current and previous versions of the Amazon ECS-optimized AMIs and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package.

The Amazon ECS-optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see [Retrieving Amazon ECS-Optimized AMI metadata \(p. 339\)](#).

Linux Amazon ECS-optimized AMIs versions

The following tabs display a list of Linux Amazon ECS-optimized AMIs versions.

Amazon Linux 2 AMI versions

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

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20211103	1.57.0	20.10.7	1.57.0-1

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20211020	1.56.0	20.10.7	1.56.0-1
20211013	1.55.5	20.10.7	1.55.5-1
20210929	1.55.4	20.10.7	1.55.4-1
20210922	1.55.3	20.10.7	1.55.3-1
20210907	1.55.2	19.03.13-ce	1.55.2-1
20210819	1.55.1	19.03.13-ce	1.55.1-1
20210805	1.55.0	19.03.13-ce	1.55.0-1
20210802	1.54.1	19.03.13-ce	1.54.1-1
20210723	1.54.1	19.03.13-ce	1.54.1-1
20210708	1.54.0	19.03.13-ce	1.54.0-1
20210623	1.53.1	19.03.13-ce	1.53.1-1
20210609	1.53.0	19.03.13-ce	1.53.0-1
20210520	1.52.2	19.03.13-ce	1.52.2-1
20210514	1.52.1	19.03.13-ce	1.52.1-1
20210504	1.52.0	19.03.13-ce	1.52.0-1
20210428	1.52.0	19.03.13-ce	1.52.0-1
20210413	1.51.0	19.03.13-ce	1.51.0-1
20210331	1.51.0	19.03.13-ce	1.51.0-1
20210316	1.50.3	19.03.13-ce	1.50.3-1
20210301	1.50.2	19.03.13-ce	1.50.2-1
20210219	1.50.2	19.03.13-ce	1.50.2-1
20210210	1.50.1	19.03.13-ce	1.50.1-1
20210202	1.50.0	19.03.13-ce	1.50.0-1
20210121	1.50.0	19.03.13-ce	1.50.0-1
20210106	1.49.0	19.03.13-ce	1.49.0-1
20201209	1.48.1	19.03.13-ce	1.48.1-1
20201130	1.48.1	19.03.13-ce	1.48.1-1
20201125	1.48.1	19.03.6-ce	1.48.1-1
20201119	1.48.0	19.03.6-ce	1.48.0-1
20201028	1.47.0	19.03.6-ce	1.47.0-1

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20201013	1.46.0	19.03.6-ce	1.46.0-1
20200928	1.45.0	19.03.6-ce	1.45.0-1
20200915	1.44.4	19.03.6-ce	1.44.4-1
20200905	1.44.3	19.03.6-ce	1.44.3-1
20200902	1.44.3	19.03.6-ce	1.44.3-1
20200827	1.44.2	19.03.6-ce	1.44.2-1
20200820	1.44.1	19.03.6-ce	1.44.1-1
20200813	1.44.0	19.03.6-ce	1.44.0-1
20200805	1.43.0	19.03.6-ce	1.43.0-1
20200723	1.42.0	19.03.6-ce	1.42.0-1
20200708	1.41.1	19.03.6-ce	1.41.1-2
20200706	1.41.1	19.03.6-ce	1.41.1-1
20200623	1.41.0	19.03.6-ce	1.41.0-1
20200603	1.40.0	19.03.6-ce	1.40.0-1
20200430	1.39.0	19.03.6-ce	1.39.0-1
20200402	1.39.0	18.09.9-ce	1.39.0-1
20200319	1.38.0	18.09.9-ce	1.38.0-1
20200218	1.37.0	18.09.9-ce	1.37.0-2
20200205	1.36.2	18.09.9-ce	1.36.2-1
20200115	1.36.1	18.09.9-ce	1.36.1-1
20200108	1.36.0	18.09.9-ce	1.36.0-1
20191212	1.35.0	18.09.9-ce	1.35.0-1
20191114	1.33.0	18.06.1-ce	1.33.0-1
20191031	1.32.1	18.06.1-ce	1.32.1-1
20191014	1.32.0	18.06.1-ce	1.32.0-1
20190925	1.32.0	18.06.1-ce	1.32.0-1
20190913	1.31.0	18.06.1-ce	1.31.0-1
20190815	1.30.0	18.06.1-ce	1.30.0-1
20190709	1.29.1	18.06.1-ce	1.29.1-1
20190614	1.29.0	18.06.1-ce	1.29.0-1

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20190607	1.29.0	18.06.1-ce	1.29.0-1
20190603	1.28.1	18.06.1-ce	1.28.1-2
20190510	1.28.0	18.06.1-ce	1.28.0-1
20190402	1.27.0	18.06.1-ce	1.27.0-1
20190301	1.26.0	18.06.1-ce	1.26.0-1
20190215	1.25.3	18.06.1-ce	1.25.3-1
20190204	1.25.2	18.06.1-ce	1.25.2-1
20190127	1.25.1	18.06.1-ce	1.25.1-1
20190118	1.25.0	18.06.1-ce	1.25.0-1
20190107	1.24.0	18.06.1-ce	1.24.0-1
20181112	1.22.0	18.06.1-ce	1.22.0-1
20181016	1.20.3	18.06.1-ce	1.21.0-1

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

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

Amazon Linux 2 (arm64) AMI versions

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

Amazon ECS-optimized Amazon Linux 2 (arm64) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20211103	1.57.0	20.10.7	1.57.0-1
20211020	1.56.0	20.10.7	1.56.0-1
20211013	1.55.5	20.10.7	1.55.5-1
20210929	1.55.4	20.10.7	1.55.4-1
20210922	1.55.3	20.10.7	1.55.3-1
20210907	1.55.2	19.03.13-ce	1.55.2-1
20210819	1.55.1	19.03.13-ce	1.55.1-1
20210805	1.55.0	19.03.13-ce	1.55.0-1

Amazon ECS-optimized Amazon Linux 2 (arm64) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20210802	1.54.1	19.03.13-ce	1.54.1-1
20210723	1.54.1	19.03.13-ce	1.54.1-1
20210708	1.54.0	19.03.13-ce	1.54.0-1
20210623	1.53.1	19.03.13-ce	1.53.1-1
20210609	1.53.0	19.03.13-ce	1.53.0-1
20210520	1.52.2	19.03.13-ce	1.52.2-1
20210514	1.52.1	19.03.13-ce	1.52.1-1
20210504	1.52.0	19.03.13-ce	1.52.0-1
20210428	1.52.0	19.03.13-ce	1.52.0-1
20210413	1.51.0	19.03.13-ce	1.51.0-1
20210331	1.51.0	19.03.13-ce	1.51.0-1
20210316	1.50.3	19.03.13-ce	1.50.3-1
20210301	1.50.2	19.03.13-ce	1.50.2-1
20210219	1.50.2	19.03.13-ce	1.50.2-1
20210210	1.50.1	19.03.13-ce	1.50.1-1
20210202	1.50.0	19.03.13-ce	1.50.0-1
20210121	1.50.0	19.03.13-ce	1.50.0-1
20210106	1.49.0	19.03.13-ce	1.49.0-1
20201209	1.48.1	19.03.13-ce	1.48.1-1
20201130	1.48.1	19.03.13-ce	1.48.1-1
20201125	1.48.1	19.03.6-ce	1.48.1-1
20201119	1.48.0	19.03.6-ce	1.48.0-1
20201028	1.47.0	19.03.6-ce	1.47.0-1
20201013	1.46.0	19.03.6-ce	1.46.0-1
20200928	1.45.0	19.03.6-ce	1.45.0-1
20200915	1.44.4	19.03.6-ce	1.44.4-1
20200905	1.44.3	19.03.6-ce	1.44.3-1
20200902	1.44.3	19.03.6-ce	1.44.3-1
20200827	1.44.2	19.03.6-ce	1.44.2-1
20200820	1.44.1	19.03.6-ce	1.44.1-1

Amazon ECS-optimized Amazon Linux 2 (arm64) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20200813	1.44.0	19.03.6-ce	1.44.0-1
20200805	1.43.0	19.03.6-ce	1.43.0-1
20200723	1.42.0	19.03.6-ce	1.42.0-1
20200708	1.41.1	19.03.6-ce	1.41.1-2
20200706	1.41.1	19.03.6-ce	1.41.1-1
20200623	1.41.0	19.03.6-ce	1.41.0-1
20200603	1.40.0	19.03.6-ce	1.40.0-1
20200430	1.39.0	19.03.6-ce	1.39.0-1
20200402	1.39.0	18.09.9-ce	1.39.0-1
20200319	1.38.0	18.09.9-ce	1.38.0-1
20200218	1.37.0	18.09.9-ce	1.37.0-2
20200205	1.36.2	18.09.9-ce	1.36.2-1
20200115	1.36.1	18.09.9-ce	1.36.1-1
20200108	1.36.0	18.09.9-ce	1.36.0-1
20191212	1.35.0	18.09.9-ce	1.35.0-1
20191114	1.33.0	18.06.1-ce	1.33.0-1
20191031	1.32.1	18.06.1-ce	1.32.1-1
20191014	1.32.0	18.06.1-ce	1.32.0-1
20190925	1.32.0	18.06.1-ce	1.32.0-1
20190913	1.31.0	18.06.1-ce	1.31.0-1
20190815	1.30.0	18.06.1-ce	1.30.0-1
20190709	1.29.1	18.06.1-ce	1.29.1-1
20190617	1.29.0	18.06.1-ce	1.29.0-1
20190607	1.29.0	18.06.1-ce	1.29.0-1
20190603	1.28.1	18.06.1-ce	1.28.1-2
20190510	1.28.0	18.06.1-ce	1.28.0-1
20190403	1.27.0	18.06.1-ce	1.27.0-1
20190301	1.26.0	18.06.1-ce	1.26.0-1
20190215	1.25.3	18.06.1-ce	1.25.3-1
20190204	1.25.2	18.06.1-ce	1.25.2-1

Amazon ECS-optimized Amazon Linux 2 (arm64) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20190127	1.25.1	18.06.1-ce	1.25.1-1
20190119	1.25.0	18.06.1-ce	1.25.0-1
20181120	1.22.0	18.06.1-ce	1.22.0-1

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

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

Amazon Linux 2 (GPU) AMI versions

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

Amazon ECS GPU-optimized AMI	Amazon ECS container agent version	Docker version	ecs-init version	NVIDIA driver version
20211103	1.57.0	20.10.7	1.57.0-1	460.73.01
20211020	1.56.0	20.10.7	1.56.0-1	460.73.01
20211013	1.55.5	20.10.7	1.55.5-1	460.73.01
20210929	1.55.4	20.10.7	1.55.4-1	460.73.01
20210923	1.55.3	20.10.7	1.55.3-1	460.73.01
20210907	1.55.2	19.03.13-ce	1.55.2-1	460.73.01
20210819	1.55.1	19.03.13-ce	1.55.1-1	460.73.01
20210805	1.55.0	19.03.13-ce	1.55.0-1	460.73.01
20210802	1.54.1	19.03.13-ce	1.54.1-1	460.73.01
20210723	1.54.1	19.03.13-ce	1.54.1-1	460.73.01
20210708	1.54.0	19.03.13-ce	1.54.0-1	460.73.01
20210623	1.53.1	19.03.13-ce	1.53.1-1	460.73.01
20210609	1.53.0	19.03.13-ce	1.53.0-1	460.73.01
20210520	1.52.2	19.03.13-ce	1.52.2-1	460.73.01
20210514	1.52.1	19.03.13-ce	1.52.1-1	460.73.01
20210504	1.52.0	19.03.13-ce	1.52.0-1	460.73.01
20210428	1.52.0	19.03.13-ce	1.52.0-1	460.73.01

Amazon ECS GPU-optimized AMI	Amazon ECS container agent version	Docker version	ecs-init version	NVIDIA driver version
20210413	1.51.0	19.03.13-ce	1.51.0-1	460.32.03
20210331	1.51.0	19.03.13-ce	1.51.0-1	460.32.03
20210316	1.50.3	19.03.13-ce	1.50.3-1	460.32.03
20210301	1.50.2	19.03.13-ce	1.50.2-1	450.51.06
20210219	1.50.2	19.03.13-ce	1.50.2-1	450.51.06
20210210	1.50.1	19.03.13-ce	1.50.1-1	450.51.06
20210202	1.50.0	19.03.13-ce	1.50.0-1	450.51.06
20210121	1.50.0	19.03.13-ce	1.50.0-1	450.51.06
20210106	1.49.0	19.03.13-ce	1.49.0-1	450.51.06
20201209	1.48.1	19.03.13-ce	1.48.1-1	450.80.02
20201130	1.48.1	19.03.13-ce	1.48.1-1	450.80.02
20201125	1.48.1	19.03.6-ce	1.48.1-1	450.80.02
20201119	1.48.0	19.03.6-ce	1.48.0-1	450.80.02
20201028	1.47.0	19.03.6-ce	1.47.0-1	450.80.02
20201013	1.46.0	19.03.6-ce	1.46.0-1	418.87.00
20200928	1.45.0	19.03.6-ce	1.45.0-1	418.87.00
20200915	1.44.4	19.03.6-ce	1.44.4-1	418.87.00
20200905	1.44.3	19.03.6-ce	1.44.3-1	418.87.00
20200902	1.44.3	19.03.6-ce	1.44.3-1	418.87.00
20200827	1.44.2	19.03.6-ce	1.44.2-1	418.87.00
20200820	1.44.1	19.03.6-ce	1.44.1-1	418.87.00
20200813	1.44.0	19.03.6-ce	1.44.0-1	418.87.00
20200805	1.43.0	19.03.6-ce	1.43.0-1	418.87.00
20200723	1.42.0	19.03.6-ce	1.42.0-1	418.87.00
20200708	1.41.1	19.03.6-ce	1.41.1-2	418.87.00
20200706	1.41.1	19.03.6-ce	1.41.1-1	418.87.00
20200623	1.41.0	19.03.6-ce	1.41.0-1	418.87.00
20200603	1.40.0	19.03.6-ce	1.40.0-1	418.87.00
20200430	1.39.0	19.03.6-ce	1.39.0-1	418.87.00
20200402	1.39.0	18.09.9-ce	1.39.0-1	418.87.00

Amazon ECS GPU-optimized AMI	Amazon ECS container agent version	Docker version	ecs-init version	NVIDIA driver version
20200319	1.38.0	18.09.9-ce	1.38.0-1	418.87.00
20200218	1.37.0	18.09.9-ce	1.37.0-2	418.87.00
20200205	1.36.2	18.09.9-ce	1.36.2-1	418.87.00
20200115	1.36.1	18.09.9-ce	1.36.1-1	418.87.00
20200108	1.36.0	18.09.9-ce	1.36.0-1	418.87.00
20191212	1.35.0	18.09.9-ce	1.35.0-1	418.87.00
20191114	1.33.0	18.06.1-ce	1.33.0-1	418.87.00
20191031	1.32.1	18.06.1-ce	1.32.1-1	418.87.00
20191014	1.32.0	18.06.1-ce	1.32.0-1	418.87.00
20190925	1.32.0	18.06.1-ce	1.32.0-1	418.87.00
20190913	1.31.0	18.06.1-ce	1.31.0-1	418.87.00
20190815	1.30.0	18.06.1-ce	1.30.0-1	418.40.04
20190709	1.29.1	18.06.1-ce	1.29.1-1	418.40.04
20190614	1.29.0	18.06.1-ce	1.29.0-1	418.40.04
20190607	1.29.0	18.06.1-ce	1.29.0-1	418.40.04
20190603	1.28.1	18.06.1-ce	1.28.1-2	418.40.04
20190510	1.28.0	18.06.1-ce	1.28.0-1	418.40.04
20190402	1.27.0	18.06.1-ce	1.27.0-1	418.40.04
20190321	1.26.0	18.06.1-ce	1.26.0-1	410.104
20190301	1.26.0	18.06.1-ce	1.26.0-1	396.26
20190215	1.25.3	18.06.1-ce	1.25.3-1	396.26
20190204	1.25.2	18.06.1-ce	1.25.2-1	396.26
20190127	1.25.1	18.06.1-ce	1.25.1-1	396.26
20190118	1.25.0	18.06.1-ce	1.25.0-1	396.26

You can retrieve the current Amazon ECS GPU-optimized AMI using the AWS CLI with the following command:

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

Amazon Linux 2 (Inferentia) AMI versions

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

Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20211103	1.57.0	20.10.7	1.57.0-1
20211020	1.56.0	20.10.7	1.56.0-1
20211013	1.55.5	20.10.7	1.55.5-1
20211020	1.56.0	20.10.7	1.56.0-1
20210929	1.55.4	20.10.7	1.55.4-1
20210922	1.55.3	20.10.7	1.55.3-1
20210907	1.55.2	19.03.13-ce	1.55.2-1
20210819	1.55.1	19.03.13-ce	1.55.1-1
20210805	1.55.0	19.03.13-ce	1.55.0-1
20210802	1.54.1	19.03.13-ce	1.54.1-1
20210723	1.54.1	19.03.13-ce	1.54.1-1
20210708	1.54.0	19.03.13-ce	1.54.0-1
20210623	1.53.1	19.03.13-ce	1.53.1-1
20210609	1.53.0	19.03.13-ce	1.53.0-1
20210520	1.52.2	19.03.13-ce	1.52.2-1
20210514	1.52.1	19.03.13-ce	1.52.1-1
20210504	1.52.0	19.03.13-ce	1.52.0-1
20210428	1.52.0	19.03.13-ce	1.52.0-1
20210413	1.51.0	19.03.13-ce	1.51.0-1
20210331	1.51.0	19.03.13-ce	1.51.0-1
20210316	1.50.3	19.03.13-ce	1.50.3-1
20210301	1.50.2	19.03.13-ce	1.50.2-1
20210219	1.50.2	19.03.13-ce	1.50.2-1
20210210	1.50.1	19.03.13-ce	1.50.1-1
20210202	1.50.0	19.03.13-ce	1.50.0-1
20210121	1.50.0	19.03.13-ce	1.50.0-1

Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI	Amazon ECS container agent version	Docker version	ecs-init version
20210106	1.49.0	19.03.13-ce	1.49.0-1
20201209	1.48.1	19.03.13-ce	1.48.1-1
20201130	1.48.1	19.03.13-ce	1.48.1-1
20201125	1.48.1	19.03.6-ce	1.48.1-1
20201119	1.48.0	19.03.6-ce	1.48.0-1
20201028	1.47.0	19.03.6-ce	1.47.0-1
20201013	1.46.0	19.03.6-ce	1.46.0-1
20200928	1.45.0	19.03.6-ce	1.45.0-1
20200915	1.44.4	19.03.6-ce	1.44.4-1
20200905	1.44.3	19.03.6-ce	1.44.3-1
20200902	1.44.3	19.03.6-ce	1.44.3-1
20200827	1.44.2	19.03.6-ce	1.44.2-1
20200820	1.44.1	19.03.6-ce	1.44.1-1
20200813	1.44.0	19.03.6-ce	1.44.0-1
20200805	1.43.0	19.03.6-ce	1.43.0-1
20200723	1.42.0	19.03.6-ce	1.42.0-1
20200708	1.41.1	19.03.6-ce	1.41.1-2
20200706	1.41.1	19.03.6-ce	1.41.1-1
20200623	1.41.0	19.03.6-ce	1.41.0-1

You can retrieve the current Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI using the AWS CLI with the following command:

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

Amazon Linux AMI versions

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

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

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2018.03.20210923	1.51.0	20.10.7	1.51.0-1
2018.03.20210723	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210519	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210413	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210331	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210316	1.50.3	19.03.13-ce	1.50.3-1
2018.03.20210301	1.50.2	19.03.13-ce	1.50.2-1
2018.03.20210219	1.50.2	19.03.13-ce	1.50.2-1
2018.03.20210210	1.50.1	19.03.13-ce	1.50.1-1
2018.03.20210202	1.50.0	19.03.13-ce	1.50.0-1
2018.03.20210121	1.50.0	19.03.13-ce	1.50.0-1
2018.03.20210106	1.49.0	19.03.13-ce	1.49.0-1
2018.03.20201209	1.48.1	19.03.13-ce	1.48.1-1
2018.03.20201130	1.48.1	19.03.13-ce	1.48.1-1
2018.03.20201125	1.48.1	19.03.6-ce	1.48.1-1
2018.03.20201119	1.48.0	19.03.6-ce	1.48.0-1
2018.03.20201028	1.47.0	19.03.6-ce	1.47.0-1
2018.03.20201013	1.46.0	19.03.6-ce	1.46.0-1
2018.03.20200928	1.45.0	19.03.6-ce	1.45.0-1
2018.03.20200915	1.44.4	19.03.6-ce	1.44.4-1
2018.03.20200905	1.44.3	19.03.6-ce	1.44.3-1
2018.03.20200902	1.44.3	19.03.6-ce	1.44.3-1
2018.03.20200827	1.44.2	19.03.6-ce	1.44.2-1
2018.03.20200820	1.44.1	19.03.6-ce	1.44.1-1
2018.03.20200813	1.44.0	19.03.6-ce	1.44.0-1
2018.03.20200805	1.43.0	19.03.6-ce	1.43.0-1
2018.03.20200723	1.42.0	19.03.6-ce	1.42.0-1
2018.03.20200708	1.41.1	19.03.6-ce	1.41.1-2
2018.03.20200706	1.41.1	19.03.6-ce	1.41.1-1
2018.03.20200623	1.41.0	19.03.6-ce	1.41.0-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2018.03.20200603	1.40.0	19.03.6-ce	1.40.0-1
2018.03.20200430	1.39.0	19.03.6-ce	1.39.0-1
2018.03.20200402	1.39.0	18.09.9-ce	1.39.0-1
2018.03.20200319	1.38.0	18.09.9-ce	1.38.0-1
2018.03.20200218	1.37.0	18.09.9-ce	1.37.0-2
2018.03.20200205	1.36.2	18.09.9-ce	1.36.2-1
2018.03.20200115	1.36.1	18.09.9-ce	1.36.1-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20191212	1.35.0	18.09.9-ce	1.35.0-1
2018.03.20191114	1.33.0	18.06.1-ce	1.33.0-1
2018.03.20191031	1.32.1	18.06.1-ce	1.32.1-1
2018.03.20191016	1.32.0	18.06.1-ce	1.32.0-1
2018.03.20191014	1.32.0	18.06.1-ce	1.32.0-1
2018.03.y	1.32.0	18.06.1-ce	1.32.0-1
2018.03.x	1.31.0	18.06.1-ce	1.31.0-1
2018.03.w	1.30.0	18.06.1-ce	1.30.0-1
2018.03.v	1.29.1	18.06.1-ce	1.29.1-1
2018.03.u	1.29.0	18.06.1-ce	1.29.0-1
2018.03.t	1.29.0	18.06.1-ce	1.29.0-1
2018.03.s	1.28.1	18.06.1-ce	1.28.1-2
2018.03.q	1.28.0	18.06.1-ce	1.28.0-1
2018.03.p	1.27.0	18.06.1-ce	1.27.0-1
2018.03.o	1.26.0	18.06.1-ce	1.26.0-1
2018.03.n	1.25.3	18.06.1-ce	1.25.3-1
2018.03.m	1.25.2	18.06.1-ce	1.25.2-1
2018.03.l	1.25.1	18.06.1-ce	1.25.1-1
2018.03.k	1.25.0	18.06.1-ce	1.25.0-1
2018.03.j	1.24.0	18.06.1-ce	1.24.0-1
2018.03.i	1.22.0	18.06.1-ce	1.22.0-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2018.03.h	1.21.0	18.06.1-ce	1.21.0-1
2018.03.g	1.20.3	18.06.1-ce	1.20.3-1
2018.03.f	1.20.2	18.06.1-ce	1.20.2-1
2018.03.e	1.20.1	18.03.1-ce	1.20.1-1
2018.03.d	1.20.0	18.03.1-ce	1.20.0-1
2018.03.c	1.19.1	18.03.1-ce	1.19.1-1
2018.03.b	1.19.0	18.03.1-ce	1.19.0-1
2018.03.a	1.18.0	17.12.1-ce	1.18.0-1
2017.09.l	1.17.3	17.12.1-ce	1.17.3-1
2017.09.k	1.17.2	17.12.0-ce	1.17.2-1
2017.09.j	1.17.2	17.12.0-ce	1.17.2-1
2017.09.i	1.17.1	17.09.1-ce	1.17.1-1
2017.09.h	1.17.0	17.09.1-ce	1.17.0-2
2017.09.g	1.16.2	17.09.1-ce	1.16.2-1
2017.09.f	1.16.1	17.06.2-ce	1.16.1-1
2017.09.e	1.16.1	17.06.2-ce	1.16.1-1
2017.09.d	1.16.0	17.06.2-ce	1.16.0-1
2017.09.c	1.15.2	17.06.2-ce	1.15.1-1
2017.09.b	1.15.1	17.06.2-ce	1.15.1-1
2017.09.a	1.15.0	17.06.2-ce	1.15.0-4
2017.03.g	1.14.5	17.03.2-ce	1.14.5-1
2017.03.f	1.14.4	17.03.2-ce	1.14.4-1
2017.03.e	1.14.3	17.03.1-ce	1.14.3-1
2017.03.d	1.14.3	17.03.1-ce	1.14.3-1
2017.03.c	1.14.3	17.03.1-ce	1.14.3-1
2017.03.b	1.14.3	17.03.1-ce	1.14.3-1
2016.09.g	1.14.1	1.12.6	1.14.1-1
2016.09.f	1.14.0	1.12.6	1.14.0-2
2016.09.e	1.14.0	1.12.6	1.14.0-1
2016.09.d	1.13.1	1.12.6	1.13.1-2

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2016.09.c	1.13.1	1.11.2	1.13.1-1
2016.09.b	1.13.1	1.11.2	1.13.1-1
2016.09.a	1.13.0	1.11.2	1.13.0-1
2016.03.j	1.13.0	1.11.2	1.13.0-1
2016.03.i	1.12.2	1.11.2	1.12.2-1
2016.03.h	1.12.1	1.11.2	1.12.1-1
2016.03.g	1.12.0	1.11.2	1.12.0-1
2016.03.f	1.11.1	1.11.2	1.11.1-1
2016.03.e	1.11.0	1.11.2	1.11.0-1
2016.03.d	1.10.0	1.11.1	1.10.0-1
2016.03.c	1.10.0	1.11.1	1.10.0-1
2016.03.b	1.9.0	1.9.1	1.9.0-1
2016.03.a	1.8.2	1.9.1	1.8.2-1
2015.09.g	1.8.1	1.9.1	1.8.1-1
2015.09.f	1.8.0	1.9.1	1.8.0-1
2015.09.e	1.7.1	1.9.1	1.7.1-1
2015.09.d	1.7.1	1.9.1	1.7.1-1
2015.09.c	1.7.0	1.7.1	1.7.0-1
2015.09.b	1.6.0	1.7.1	1.6.0-1
2015.09.a	1.5.0	1.7.1	1.5.0-1
2015.03.g	1.4.0	1.7.1	1.4.0-2
2015.03.f	1.4.0	1.6.2	1.4.0-1
2015.03.e	1.3.1	1.6.2	1.3.1-1
2015.03.d	1.2.1	1.6.2	1.2.0-2
2015.03.c	1.2.0	1.6.2	1.2.0-1
2015.03.b	1.1.0	1.6.0	1.0-3
2015.03.a	1.0.0	1.5.0	1.0-1

You can retrieve the current Amazon ECS-optimized Amazon Linux AMI using the AWS CLI with the following command:

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

Additional resources

The following pages provide additional information about the changes:

- [Amazon ECS changelog](#) on GitHub
- [Amazon ecs-init changelog](#) on GitHub
- [Docker Engine release notes](#) in the Docker documentation
- [NVIDIA Driver Documentation](#) in the NVIDIA documentation

AMI storage configuration

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

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

Topics

- [Amazon Linux 2 storage configuration \(p. 357\)](#)
- [Amazon ECS-optimized Amazon Linux AMI storage configuration \(p. 357\)](#)

Amazon Linux 2 storage configuration

By default, the Amazon Linux 2-based Amazon ECS-optimized AMIs (Amazon ECS-optimized Amazon Linux 2 AMI, Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, and Amazon ECS GPU-optimized AMI) ship with a single 30-GiB root volume. You can modify the 30-GiB root volume size at launch time to increase the available storage on your container instance. This storage is used for the operating system and for Docker images and metadata.

The default filesystem for the Amazon ECS-optimized Amazon Linux 2 AMI is `ext4`, and Docker uses the `overlay2` storage driver. For more information, see [Use the OverlayFS storage driver](#) in the Docker documentation.

Amazon ECS-optimized Amazon Linux AMI storage configuration

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

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

Version 2015.09.d and later

Amazon ECS-optimized Amazon Linux AMIs from version 2015.09.d and later launch with an 8-GiB volume for the operating system that is attached at `/dev/xvda` and mounted as the root of the file system. There is an additional 22-GiB volume that is attached at `/dev/xvdcz` that Docker uses for image and metadata storage. The volume is configured as a Logical Volume Management (LVM)

device and it is accessed directly by Docker via the devicemapper backend. Because the volume is not mounted, you cannot use standard storage information commands (such as `df -h`) to determine the available storage. However, you can use LVM commands and `docker info` to find the available storage by following the procedure below. For more information, see the [LVM HOWTO](#) in The Linux Documentation Project.

Note

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

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

Note

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

To determine the available storage for Docker

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

Note

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

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

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

Output:

```
VG     #PV #LV #SN Attr   VSize  VFree
docker  1    1    0 wz--n- 22.00g 204.00m
```

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

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

Output:

```
LV        VG        Attr      LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync
Convert
docker-pool docker twi-aot--- 21.75g                  7.63   4.96
```

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

Note

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

```
[ec2-user ~]$ docker info | grep "Data Space"
```

Output:

```
Data Space Used: 1.782 GB
Data Space Total: 23.35 GB
Data Space Available: 21.57 GB
```

To extend the Docker logical volume

The easiest way to add storage to your container instances is to terminate the existing instances and launch new ones with larger data storage volumes. However, if you are unable to do this, you can add storage to the volume group that Docker uses and extend its logical volume by following these steps.

Note

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

- (Amazon ECS container agent 1.8.0 and later) Reduce the amount of time that stopped or exited containers remain on your container instances. The `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` agent configuration variable sets the time duration to wait from when a task is stopped until the Docker container is removed (by default, this value is 3 hours). This removes the Docker container data. If this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).
- Remove non-running containers and unused images from your container instances. You can use the following example commands to manually remove stopped containers and unused images. Deleted containers cannot be inspected later, and deleted images must be pulled again before starting new containers from them.

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

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

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

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

- Remove unused data blocks within containers. You can use the following command to run `fstrim` on any running container and discard any data blocks that are unused by the container file system.

```
$ sudo sh -c "docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -IZ fstrim /proc/Z/root/"
```

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

2. Attach the volume to your container instance. The default location for the Docker data volume is `/dev/xvdcz`. For consistency, attach additional volumes in reverse alphabetical order from that device name (for example, `/dev/xvdcy`). For more information, see [Attaching an Amazon EBS Volume to an Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.
3. Connect to your container instance using SSH. For more information, see [Connect to your container instance \(p. 384\)](#).
4. Check the size of your `docker-pool` logical volume. The example below shows a logical volume of 409.19 GiB.

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

Output:

LV	VG	Attr	LSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%	Sync
Convert											
docker-pool	docker	twi-aot---	409.19g					0.16	0.08		

5. Check the current available space in your volume group. The example below shows 612.75 GiB in the `VFree` column.

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

Output:

VG	#PV	#LV	#SN	Attr	VSize	VFree
docker	1	1	0	wz--n-	1024.00g	612.75g

6. Add the new volume to the `docker` volume group, substituting the device name to which you attached the new volume. In this example, a 1-TiB volume was previously added and attached to `/dev/xvdcy`.

```
[ec2-user ~]$ sudo vgextend docker /dev/xvdcy
Physical volume "/dev/sdycy" successfully created
Volume group "docker" successfully extended
```

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

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

Output:

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

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

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

Output:

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

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

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

Output:

LV	VG	Attr	lSize	Pool	Origin	Data%	Meta%	Move	Log	Cpy%Sync
Convert										
docker-pool	docker	twi-aot---	1.40t			0.04	0.12			

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

Note

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

```
[ec2-user ~]$ docker info | grep "Data Space"
```

Output:

```
Data Space Used: 109.6 MB
Data Space Total: 1.539 TB
Data Space Available: 1.539 TB
```

Version 2015.09.c and earlier

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

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

Amazon ECS-optimized Linux AMI build script

Amazon ECS has open-sourced the build scripts that are used to build the Linux variants of the Amazon ECS-optimized AMI. These build scripts are now available on GitHub. For more information, see [amazon-ecs-ami](#) on GitHub.

The build scripts repository includes a [HashiCorp packer](#) template and build scripts to generate each of the Linux variants of the Amazon ECS-optimized AMI. These scripts are the source of truth for Amazon ECS-optimized AMI builds, so you can follow the GitHub repository to monitor changes to our AMIs. For example, perhaps you want your own AMI to use the same version of Docker that the Amazon ECS team uses for the official AMI.

For more information, see the Amazon ECS AMI repository at [aws/amazon-ecs-ami](#) on GitHub.

To build an Amazon ECS-optimized Linux AMI

1. Clone the `aws/amazon-ecs-ami` GitHub repo.

```
git clone git@github.com:aws/amazon-ecs-ami.git
```

2. Add an environment variable for the AWS Region to use when creating the AMI. Replace the `us-west-2` value with the Region to use.

```
export REGION=us-west-2
```

3. A Makefile is provided to build the AMI. From the root directory of the cloned repository, use one of the following commands, corresponding to the Linux variant of the Amazon ECS-optimized AMI you want to build.

- Amazon ECS-optimized Amazon Linux 2 AMI

```
make al2
```

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

```
make al2arm
```

- Amazon ECS GPU-optimized AMI

```
make al2gpu
```

- Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI

```
make al2inf
```

- Amazon ECS-optimized Amazon Linux AMI

```
make al1
```

Important

On April 15, 2021, the Amazon ECS-optimized Amazon Linux AMI ended its standard support phase and entered a maintenance support phase. In the maintenance support phase, Amazon ECS will continue providing critical and important security updates for a reduced list of packages. During this period, Amazon ECS will no longer add support for new EC2 instance types, new services and features, and new packages. Instead, Amazon ECS will provide updates only for critical and important security fixes that apply to a reduced set of packages. Maintenance support period will end on June 30, 2023.

Using Bottlerocket with Amazon ECS

Bottlerocket is a Linux-based open source operating system that is purpose-built by AWS for running containers. For more information, see [Bottlerocket on GitHub](#).

An Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI you can use when launching Amazon ECS container instances. For a detailed walkthrough of how to get started with the Bottlerocket operating system on Amazon ECS, see [Using a Bottlerocket AMI with Amazon ECS](#).

You can request new features on the GitHub page. For more information, see [Bottlerocket on GitHub](#).

Considerations

The following should be considered when using the Bottlerocket AMI with Amazon ECS.

- Bottlerocket is optimized for container workloads and has a focus on security. It does not include a package manager and is immutable by default. For information about the security features and guidance, see [Security Features](#) and [Security Guidance](#) on the GitHub website.
- The Amazon ECS variant of the Bottlerocket AMI is not supported in the following Regions:
 - China (Beijing) (cn-north-1)
 - China (Ningxia) (cn-northwest-1)
- Amazon EC2 instances with x86 or arm64 processors are supported. Amazon EC2 instances with GPUs or Inferentia chips are not supported.
- The `awsvpc` network mode is supported when using Bottlerocket AMI version 1.1.0 or later.
- Using Amazon EFS file system volumes are not supported.
- The `initProcessEnabled` task definition parameter is not supported.
- The following features are not supported:
 - App Mesh in task definitions
 - ECS Anywhere
 - ECS Exec
 - Amazon EFS in encrypted mode and `awsvpc` network mode
 - Elastic Inference
 - FireLens in task definitions

The Amazon ECS variant of the Bottlerocket AMI can be retrieved using a Systems Manager parameter. The following is the format of the parameter name.

```
/aws/service/bottlerocket/aws-ecs-1/x86_64/latest
```

A specific version of the Bottlerocket AMI can be retrieved by using the version number in place of the latest tag. The following is an example.

```
aws ssm get-parameters --name "/aws/service/bottlerocket/aws-ecs-1/x86_64/1.1.0/image_id"
--region us-east-1
```

You can retrieve the latest stable Bottlerocket AMI using the AWS CLI with the following command.

```
aws ssm get-parameters --name "/aws/service/bottlerocket/aws-ecs-1/x86_64/latest/image_id"
--region us-east-1
```

The following table provides a link to retrieve the latest AMI ID of the Amazon ECS variant of the Bottlerocket operating system by Region.

Region Name	Region	AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US East (Ohio)	us-east-2	View AMI ID
US West (N. California)	us-west-1	View AMI ID

Region Name	Region	AMI ID
US West (Oregon)	us-west-2	View AMI ID
Africa (Cape Town)	af-south-1	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Osaka)	ap-northeast-3	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Europe (Milan)	eu-south-1	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID

Launching an Amazon ECS Linux container instance

Your Amazon ECS container instances are created using the Amazon EC2 console. Before you begin, be sure that you've completed the steps in [Setting up with Amazon ECS \(p. 7\)](#).

To launch a container instance

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select the Region to use.
3. From the **EC2 Dashboard**, choose **Launch instance**.
4. On the **Choose an Amazon Machine Image (AMI)** page, complete the following steps:
 - a. Choose **AWS Marketplace**.
 - b. Choose an AMI for your container instance. You can search for one of the Amazon ECS-optimized AMIs, for example the **Amazon ECS-Optimized Amazon Linux 2 AMI**. If you do not choose an Amazon ECS-optimized AMI, you must follow the procedures in [Installing the Amazon ECS container agent \(p. 431\)](#).

For more information on the latest Amazon ECS-optimized AMIs, see [Amazon ECS-optimized AMI \(p. 333\)](#).

Important

The **Amazon ECS-optimized Amazon Linux AMI** is deprecated as of April 15, 2021. After that date, Amazon ECS will continue providing critical and important security updates for the AMI but will not add support for new features.

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

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

6. On the **Configure Instance Details** page, complete the following steps:
 - a. Set the **Number of instances** field depending on how many container instances you want to add to your cluster.
 - b. (Optional) To use Spot Instances, for **Purchasing option**, select the check box next to **Request Spot Instances**. You also need to set the other fields related to Spot Instances. For more information, see [Spot Instance Requests](#).

Note

If you are using Spot Instances and see a `Not available` message, you may need to choose a different instance type.

- c. For **Network**, choose the VPC into which to launch your container instance.
- d. For **Subnet**, choose a subnet to use, or keep the default option to choose the default subnet in any Availability Zone.
- e. Set the **Auto-assign Public IP** field depending on whether you want your instance to be accessible from the public internet. If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

Note

Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 714\)](#).

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 504\)](#) in this guide. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters \(p. 727\)](#).

- f. Select your container instance IAM role. This is usually named `ecsInstanceRole`.

Important

If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

- g. (Optional) Configure your Amazon ECS container instance with user data, such as the agent environment variables from [Amazon ECS container agent configuration \(p. 454\)](#). Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:
 - By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing `your_cluster_name` with the name of your cluster.

```
#!/bin/bash
echo ECS_CLUSTER=your_cluster_name >> /etc/ecs/ecs.config
```

- If you have an `ecs.config` file in Amazon S3 and have enabled Amazon S3 read-only access to your container instance role, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing `your_bucket_name` with the name of your bucket to install the AWS CLI and write your configuration file at launch time.

Note

For more information about this configuration, see [Storing Container Instance Configuration in Amazon S3 \(p. 467\)](#).

```
#!/bin/bash
yum install -y aws-cli
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

- Specify tags for your container instance using the `ECS_CONTAINER_INSTANCE_TAGS` configuration parameter. This creates tags that are associated with Amazon ECS only, they cannot be listed using the Amazon EC2 API.

Important

If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the `ECS_CONTAINER_INSTANCE_TAGS` agent configuration parameter to add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- Specify tags for your container instance and then use the `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` configuration parameter to propagate them from Amazon EC2 to Amazon ECS

The following is an example of a user data script that would propagate the tags associated with a container instance, as well as register the container instance with a cluster named `your_cluster_name`:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

For more information, see [Bootstrapping container instances with Amazon EC2 user data \(p. 368\)](#).

- h. Choose **Next: Add Storage**.
7. On the **Add Storage** page, configure the storage for your container instance.

If you are using the Amazon ECS-optimized Amazon Linux 2 AMI, your instance has a single 30 GiB volume configured, which is shared between the operating system and Docker.

If you are using the Amazon ECS-optimized AMI, your instance has two volumes configured. The **Root** volume is for the operating system's use, and the second Amazon EBS volume (attached to `/dev/xvdcz`) is for Docker's use.

You can optionally increase or decrease the volume sizes for your instance to meet your application needs.

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

8. On the **Add Tags** page, specify tags by providing key and value combinations for the container instance. Choose **Add another tag** to add more than one tag to your container instance. For more information about resource tags, see [Resources and tags \(p. 604\)](#).

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

9. On the **Configure Security Group** page, use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored. Select or create a security group as follows, and then choose **Review and Launch**.
10. On the **Review Instance Launch** page, under **Security Groups**, you see that the wizard created and selected a security group for you. Instead, select the security group that you created in [Setting up with Amazon ECS \(p. 7\)](#) using the following steps:
 - a. Choose **Edit security groups**.
 - b. On the **Configure Security Group** page, select the **Select an existing security group** option.
 - c. Select the security group you created for your container instance from the list of existing security groups, and choose **Review and Launch**.
11. On the **Review Instance Launch** page, choose **Launch**.
12. In the **Select an existing key pair or create a new key pair** dialog box, choose **Choose an existing key pair**, then select the key pair that you created when getting set up.

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

13. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.
14. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is **Pending**. After the instance starts, its state changes to **Running**, and it receives a public DNS name. If the **Public DNS** column is hidden, choose **Show/Hide, Public DNS**.

Using Spot Instances

A Spot Instance is an unused Amazon EC2 instance that is available for less than the On-Demand price. Because Spot Instances enable you to request unused EC2 instances at steep discounts, you can lower your Amazon EC2 costs significantly. The hourly price for a Spot Instance is called a Spot price. The Spot price of each instance type in each Availability Zone is set by Amazon EC2, and adjusted gradually based on the long-term supply of and demand for Spot Instances. For more information, see [Spot Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

You can register Spot Instances to your Amazon ECS clusters. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

Spot Instance Draining

Amazon EC2 terminates, stops, or hibernates your Spot Instance when the Spot price exceeds the maximum price for your request or capacity is no longer available. Amazon EC2 provides a Spot Instance two-minute interruption notice for terminate and stop actions. It does not provide the two-minute notice for the hibernate action. If Amazon ECS Spot Instance draining is enabled on the instance, ECS receives the Spot Instance interruption notice and places the instance in **DRAINING** status.

Important

Amazon ECS does not receive a notice from Amazon EC2 when instances are removed by Auto Scaling Capacity Rebalancing. For more information, see [Amazon EC2 Auto Scaling Capacity Rebalancing](#).

When a container instance is set to DRAINING, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Service tasks on the draining container instance that are in the PENDING state are stopped immediately. If there are container instances in the cluster that are available, replacement service tasks are started on them.

Spot Instance draining is disabled by default and must be manually enabled. To enable Spot Instance draining for a new container instance, when launching the container instance add the following script into the **User data** field, replacing *MyCluster* with the name of the cluster to register the container instance to.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
EOF
```

For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

To enable Spot Instance draining for an existing container instance

1. Connect to the Spot Instance over SSH.
2. Edit the /etc/ecs/ecs.config file and add the following:

```
ECS_ENABLE_SPOT_INSTANCE_DRAINING=true
```

3. Restart the ecs service.
 - For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```
 - For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo stop ecs && sudo start ecs
```
4. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see [the section called “Container agent introspection” \(p. 503\)](#).

```
curl http://localhost:51678/v1/metadata
```

Bootstrapping container instances with Amazon EC2 user data

When you launch an Amazon ECS container instance, you have the option of passing user data to the instance. The data can be used to perform common automated configuration tasks and even run scripts when the instance boots. For Amazon ECS, the most common use cases for user data are to pass configuration information to the Docker daemon and the Amazon ECS container agent.

You can pass multiple types of user data to Amazon EC2, including cloud booothooks, shell scripts, and `cloud-init` directives. For more information about these and other format types, see the [Cloud-Init documentation](#).

You can pass this user data when using the Amazon EC2 launch wizard. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

Topics

- [Amazon ECS container agent \(p. 369\)](#)
- [Docker daemon \(p. 369\)](#)

Amazon ECS container agent

The Linux variants of the Amazon ECS-optimized AMI look for agent configuration data in the `/etc/ecs/ecs.config` file when the container agent starts. You can specify this configuration data at launch with Amazon EC2 user data. For more information about available Amazon ECS container agent configuration variables, see [Amazon ECS container agent configuration \(p. 454\)](#).

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

```
#!/bin/bash
echo "ECS_CLUSTER=MyCluster" >> /etc/ecs/ecs.config
```

If you have multiple variables to write to `/etc/ecs/ecs.config`, use the following heredoc format. This format writes everything between the lines beginning with `cat` and `EOF` to the configuration file.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_ENGINE_AUTH_TYPE=docker
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":
 {"username": "my_name", "password": "my_password", "email": "email@example.com"}}
ECS_LOGLEVEL=debug
EOF
```

Docker daemon

You can specify Docker daemon configuration information with Amazon EC2 user data. For more information about configuration options, see [the Docker daemon documentation](#).

In the example below, the custom options are added to the Docker daemon configuration file, `/etc/docker/daemon.json` which is then specified in the user data when the instance is launched.

```
#!/bin/bash
cat <<EOF >/etc/docker/daemon.json
{"debug": true}
EOF
systemctl restart docker --no-block
```

Starting a task at container instance launch time

Depending on your application architecture design, you may need to run a specific container on every container instance to deal with operations or security concerns such as monitoring, security, metrics, service discovery, or logging.

To do this, you can configure your container instances to call the **docker run** command with the user data script at launch, or in some init system such as Upstart or **systemd**. While this method works, it has some disadvantages because Amazon ECS has no knowledge of the container and cannot monitor the CPU, memory, ports, or any other resources used. To ensure that Amazon ECS can properly account for all task resources, create a task definition for the container to run on your container instances. Then, use Amazon ECS to place the task at launch time with Amazon EC2 user data.

The Amazon EC2 user data script in the following procedure uses the Amazon ECS introspection API to identify the container instance. Then, it uses the AWS CLI and the **start-task** command to run a specified task on itself during startup.

To start a task at container instance launch time

1. If you have not done so already, create a task definition with the container you want to run on your container instance at launch by following the procedures in [Creating a task definition using the new console \(p. 196\)](#).
2. Modify your `ecsInstanceRole` IAM role to add permissions for the `StartTask` API operation. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
 - a. Open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. In the navigation pane, choose **Roles**.
 - c. Choose the `ecsInstanceRole`. If the role does not exist, use the procedure in [Amazon ECS container instance IAM role \(p. 695\)](#) to create the role and return to this procedure. If the role does exist, select the role to view the attached policies.
 - d. In the **Permissions** tab, choose **Add inline policy**.
 - e. For **Service**, choose **Choose a service, Elastic Container Service**.
 - f. For **Actions**, type **StartTask** in the search field, and then select **StartTask**.
 - g. For **Resources**, select **All resources**, and then choose **Review policy**.
 - h. On the **Review policy** page, enter a name for your policy, such as `ecs-start-task` and choose **Create policy**.
3. Launch one or more container instances using the Amazon ECS-optimized Amazon Linux 2 AMI by following the procedure in [Launching an Amazon ECS Linux container instance \(p. 364\)](#), but in Step 6.g (p. 365) copy and paste the MIME multi-part user data script below into the **User data** field. Substitute `your_cluster_name` with the cluster for the container instance to register into and `my_task_def` with the task definition to run on the instance at launch.

Note

The MIME multi-part content below uses a shell script to set configuration values and install packages. It also uses a systemd job to start the task after the `ecs` service is running and the introspection API is available.

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

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

#!/bin/bash
# Specify the cluster that the container instance should register into
cluster=your_cluster_name

# Write the cluster configuration variable to the ecs.config file
# (add any other configuration variables here also)
echo ECS_CLUSTER=$cluster >> /etc/ecs/ecs.config

START_TASK_SCRIPT_FILE="/etc/ecs/ecs-start-task.sh"
cat <<- 'EOF' > ${START_TASK_SCRIPT_FILE}
exec 2>>/var/log/ecs/ecs-start-task.log
```

```

set -x

# Install prerequisite tools
yum install -y jq aws-cli

# Wait for the ECS service to be responsive
until curl -s http://localhost:51678/v1/metadata
do
    sleep 1
done

# Grab the container instance ARN and AWS Region from instance metadata
instance_arn=$(curl -s http://localhost:51678/v1/metadata | jq -r '.'
| .ContainerInstanceArn' | awk -F/ '{print $NF}' )
cluster=$(curl -s http://localhost:51678/v1/metadata | jq -r '. | .Cluster' | awk -F/
'{print $NF}' )
region=$(curl -s http://localhost:51678/v1/metadata | jq -r '.'
| .ContainerInstanceArn' | awk -F: '{print $4}')

# Specify the task definition to run at launch
task_definition=my_task_def

# Run the AWS CLI start-task command to start your task on this container instance
aws ecs start-task --cluster $cluster --task-definition $task_definition --container-
instances $instance_arn --started-by $instance_arn --region $region
EOF

# Write systemd unit file
UNIT="ecs-start-task.service"
cat <<- EOF > /etc/systemd/system/${UNIT}
[Unit]
Description=ECS Start Task
Requires=ecs.service
After=ecs.service

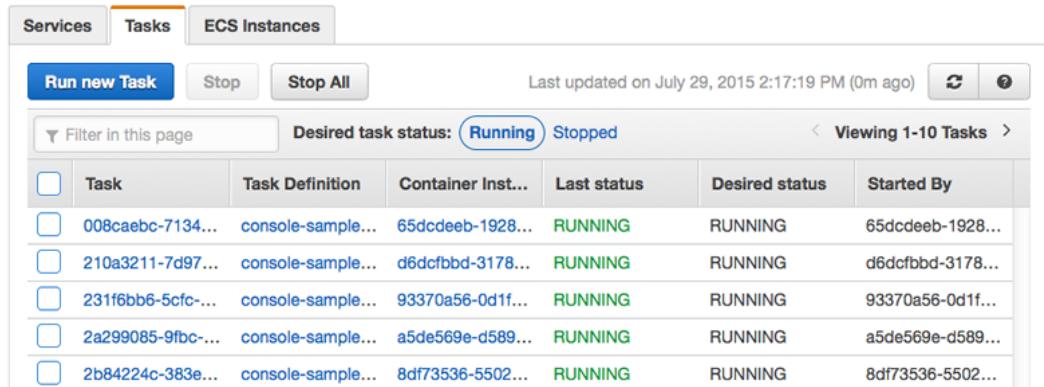
[Service]
Restart=on-failure
RestartSec=30
ExecStart=/usr/bin/bash ${START_TASK_SCRIPT_FILE}

[Install]
WantedBy=default.target
EOF

# Enable our ecs.service dependent service with `--no-block` to prevent systemd
# deadlock
# See https://github.com/aws/amazon-ecs-agent/issues/1707
systemctl enable --now --no-block "${UNIT}"
---BOUNDARY---

```

4. Verify that your container instances launch into the correct cluster and that your tasks have started.
 - a. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
 - b. From the navigation bar, choose the Region that your cluster is in.
 - c. In the navigation pane, choose **Clusters** and select the cluster that hosts your container instances.
 - d. On the **Cluster** page, choose **Tasks**.



Function Name	Metric Name	Unit	Value	Timestamp
lambda-function-1	ApproximateInvokeCount	Double	1.0	2023-01-12T12:00:00Z
lambda-function-1	ApproximateInvokeCount	Double	1.0	2023-01-12T12:00:00Z
lambda-function-1	ApproximateInvokeCount	Double	1.0	2023-01-12T12:00:00Z
lambda-function-1	ApproximateInvokeCount	Double	1.0	2023-01-12T12:00:00Z

Each container instance you launched should have your task running on it, and the container instance ARN should be in the **Started By** column.

If you do not see your tasks, you can log in to your container instances with SSH and check the `/var/log/ecs/ecs-start-task.log` file for debugging information.

Elastic network interface trunking

Note

This feature is not available on Window containers on Fargate.

Each Amazon ECS task that uses the `awsvpc` network mode receives its own elastic network interface (ENI), which is attached to the container instance that hosts it. There is a default limit to the number of network interfaces that can be attached to an Amazon EC2 instance, and the primary network interface counts as one. For example, by default a `c5.1.large` instance may have up to three ENIs attached to it. The primary network interface for the instance counts as one, so you can attach an additional two ENIs to the instance. Because each task using the `awsvpc` network mode requires an ENI, you can typically only run two such tasks on this instance type.

Amazon ECS supports launching container instances with increased ENI density using supported Amazon EC2 instance types. When you use these instance types and opt in to the `awsvpcTrunking` account setting, additional ENIs are available on newly launched container instances. This configuration allows you to place more tasks using the `awsvpc` network mode on each container instance. Using this feature, a `c5.1.large` instance with `awsvpcTrunking` enabled has an increased ENI limit of twelve. The container instance will have the primary network interface and Amazon ECS creates and attaches a "trunk" network interface to the container instance. So this configuration allows you to launch ten tasks on the container instance instead of the current two tasks.

The trunk network interface is fully managed by Amazon ECS and is deleted when you either terminate or deregister your container instance from the cluster. For more information, see [Amazon ECS task networking \(p. 278\)](#).

ENI trunking considerations

There are several things to consider when using the ENI trunking feature.

- Only Linux variants of the Amazon ECS-optimized AMI, or other Amazon Linux variants with version `1.28.1` or later of the container agent and version `1.28.1-2` or later of the `ecs-init` package, support the increased ENI limits. If you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. Windows containers are not supported at this time.

- Only new Amazon EC2 instances launched after opting in to `awsvpcTrunking` receive the increased ENI limits and the trunk network interface. Previously launched instances do not receive these features regardless of the actions taken.
- Amazon EC2 instances in shared subnets are not supported. They will fail to register to a cluster if they are used.
- Your Amazon ECS tasks must use the `awsvpc` network mode and the EC2 launch type. Tasks using the Fargate launch type always received a dedicated ENI regardless of how many are launched, so this feature is not needed.
- Your Amazon ECS tasks must be launched in the same Amazon VPC as your container instance. Your tasks will fail to start with an attribute error if they are not within the same VPC.
- When launching a new container instance, the instance transitions to a `REGISTERING` status while the trunk elastic network interface is provisioned for the instance. If the registration fails, the instance transitions to a `REGISTRATION_FAILED` status. You can troubleshoot a failed registration by describing the container instance to view the `statusReason` field which describes the reason for the failure. The container instance then can be manually deregistered or terminated. Once the container instance is successfully deregistered or terminated, Amazon ECS will delete the trunk ENI.

Note

Amazon ECS emits container instance state change events which you can monitor for instances that transition to a `REGISTRATION_FAILED` state. For more information, see [Container instance state change events \(p. 633\)](#).

- Once the container instance is terminated, the instance transitions to a `Deregistering` status while the trunk elastic network interface is deprovisioned. The instance then transitions to an `Inactive` status.
- If a container instance in a public subnet with the increased ENI limits is stopped and then restarted, the instance loses its public IP address, and the container agent loses its connection.

Working with container instances with increased ENI limits

Before you launch a container instance with the increased ENI limits, the following prerequisites must be completed.

- The service-linked role for Amazon ECS must be created. The Amazon ECS service-linked role provides Amazon ECS with the permissions to make calls to other AWS services on your behalf. This role is created for you automatically when you create a cluster, or if you create or update a service in the AWS Management Console. For more information, see [Service-linked role for Amazon ECS \(p. 685\)](#). You can also create the service-linked role with the following AWS CLI command.

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

- Your account or container instance IAM role must opt-in to the `awsvpcTrunking` account setting. This can be done in the following ways:
 - Any user can use the `PutAccountSettingDefault` API to opt-in all IAM users and roles on an account
 - A root user can use the `PutAccountSetting` API to opt-in the IAM user or container instance role that will register the instance with the cluster
 - A container instance role can opt itself in when the `PutAccountSetting` API is run on an instance prior to it being registered with a cluster

For more information, see [Account settings \(p. 324\)](#).

Once the prerequisites are met, you can launch a new container instance using one of the supported Amazon EC2 instance types, and the instance will have the increased ENI limits. For a list of supported instance types, see [Supported Amazon EC2 instance types \(p. 375\)](#). The container instance must have version 1.28.1 or later of the container agent and version 1.28.1-2 or later of the `ecs-init` package. If

you use the latest Linux variant of the Amazon ECS-optimized AMI, these requirements will be met. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

To opt in all IAM users or roles on your account to the increased ENI limits using the console

1. As the root user of the account, open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation bar at the top of the screen, select the Region for which to opt in to the increased ENI limits.
3. From the dashboard, choose **Account Settings**.
4. For **IAM user or role**, ensure your root user or container instance IAM role is selected.
5. For **AWSVPC Trunking**, select the check box. Choose **Save** once finished.

Important

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

6. On the confirmation screen, choose **Confirm** to save the selection.

To opt in all IAM users or roles on your account to the increased ENI limits using the command line

Any user on an account can use one of the following commands to modify the default account setting for all IAM users or roles on your account. These changes apply to the entire AWS account unless an IAM user or role explicitly overrides these settings for themselves.

- [put-account-setting-default \(AWS CLI\)](#)

```
aws ecs put-account-setting-default \
    --name awsvpcTrunking \
    --value enabled \
    --region us-east-1
```

- [Write-ECSAccountSettingDefault \(AWS Tools for Windows PowerShell\)](#)

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

To opt in an IAM user or container instance IAM role to the increased ENI limits as the root user using the command line

The root user on an account can use one of the following commands and specify the ARN of the principal IAM user or container instance IAM role in the request to modify the account settings.

- [put-account-setting \(AWS CLI\)](#)

The following example is for modifying the account setting of a specific IAM user:

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

The following example is for modifying the account setting of a specific container instance IAM role:

```
aws ecs put-account-setting \
--name awsvpcTrunking \
--value enabled \
--principal-arn arn:aws:iam::aws_account_id:role/ecsInstanceRole \
--region us-east-1
```

- [Write-ECSAccountSetting](#) (AWS Tools for Windows PowerShell)

The following example is for modifying the account setting of a specific IAM user:

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

The following example is for modifying the account setting of a specific container instance IAM role:

```
Write-ECSAccountSetting -Name awsvpcTrunking -Value enabled -PrincipalArn
arn:aws:iam::aws_account_id:role/ecsInstanceRole -Region us-east-1 -Force
```

To view your container instances with increased ENI limits with the AWS CLI

Each container instance has a default network interface, referred to as a trunk network interface. Use the following command to list your container instances with increased ENI limits by querying for the `ecs.awsVpcTrunkId` attribute, which indicates it has a trunk network interface.

- [list-attributes](#) (AWS CLI)

```
aws ecs list-attributes \
--target-type container-instance \
--attribute-name ecs.awsVpcTrunkId \
--cluster cluster_name \
--region us-east-1
```

- [Get-ECSAttributeList](#) (AWS Tools for Windows PowerShell)

```
Get-ECSAttributeList -TargetType container-instance -AttributeName ecs.awsVpcTrunkId -
Region us-east-1
```

Supported Amazon EC2 instance types

The following shows the supported Amazon EC2 instance types and how many tasks using the awsvpc network mode can be launched on each instance type before and after opting in to the awsvpcTrunking account setting. For the elastic network interface (ENI) limits on each instance type, add one to the current task limit, as the primary network interface counts against the limit, and add two to the new task limit, as both the primary network interface and the trunk network instance count again the limit.

Important

Although other instance types are supported in the same instance family, the c5n, m5n, m5dn, r5n, and r5dn instance types are not supported.

a1 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
a1.medium	1	10
a1.large	2	10
a1.xlarge	3	20
a1.2xlarge	3	40
a1.4xlarge	7	60

c5 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
c5.large	2	10
c5.xlarge	3	20
c5.2xlarge	3	40
c5.4xlarge	7	60
c5.9xlarge	7	60
c5.12xlarge	7	60
c5.18xlarge	14	120
c5.24xlarge	14	120
c5a.large	2	10
c5a.xlarge	3	20
c5a.2xlarge	3	40
c5a.4xlarge	7	60
c5a.12xlarge	7	60
c5a.16xlarge	14	120
c5a.18xlarge	14	120
c5a.24xlarge	14	120
c5ad.large	2	10
c5ad.xlarge	3	20
c5ad.2xlarge	3	40
c5ad.4xlarge	7	60

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
c5ad.12xlarge	7	60
c5ad.16xlarge	14	120
c5ad.18xlarge	14	120
c5ad.24xlarge	14	120
c5d.large	2	10
c5d.xlarge	3	20
c5d.2xlarge	3	40
c5d.4xlarge	7	60
c5d.9xlarge	7	60
c5d.12xlarge	7	60
c5d.18xlarge	14	120
c5d.24xlarge	14	120

c6 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
c6g.medium	1	4
c6g.large	2	10
c6g.xlarge	3	20
c6g.2xlarge	3	40
c6g.4xlarge	7	60
c6g.8xlarge	7	60
c6g.12xlarge	7	60
c6g.16xlarge	14	120
c6g.metal	14	120
c6gd.medium	1	4
c6gd.large	2	10
c6gd.xlarge	3	20
c6gd.2xlarge	3	40
c6gd.4xlarge	7	60
c6gd.8xlarge	7	60

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
c6gd.12xlarge	7	60
c6gd.16xlarge	14	120
c6gd.metal	14	120

g3 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
g3.4xlarge	7	12
g3.8xlarge	7	12
g3.16xlarge	14	12
g3s.xlarge	3	12

m5 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
m5.large	2	10
m5.xlarge	3	20
m5.2xlarge	3	40
m5.4xlarge	7	60
m5.8xlarge	7	60
m5.12xlarge	7	60
m5.16xlarge	14	120
m5.24xlarge	14	120
m5a.large	2	10
m5a.xlarge	3	20
m5a.2xlarge	3	40
m5a.4xlarge	7	60
m5a.8xlarge	7	60
m5a.12xlarge	7	60
m5a.16xlarge	14	120
m5a.24xlarge	14	120

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
m5ad.large	2	10
m5ad.xlarge	3	20
m5ad.2xlarge	3	40
m5ad.4xlarge	7	60
m5ad.8xlarge	7	60
m5ad.12xlarge	7	60
m5ad.16xlarge	14	120
m5ad.24xlarge	14	120
m5d.large	2	10
m5d.xlarge	3	20
m5d.2xlarge	3	40
m5d.4xlarge	7	60
m5d.8xlarge	7	60
m5d.12xlarge	7	60
m5d.16xlarge	14	120
m5d.24xlarge	14	120
m5d.metal	14	120

m6 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
m6i.large	2	10
m6i.xlarge	3	20
m6i.2xlarge	3	40
m6i.4xlarge	7	60
m6i.8xlarge	7	90
m6i.12xlarge	7	120
m6i.16xlarge	14	120
m6i.24xlarge	14	120
m6i.32xlarge	14	120
m6g.medium	1	4

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
m6g.large	2	10
m6g.xlarge	3	20
m6g.2xlarge	3	40
m6g.4xlarge	7	60
m6g.8xlarge	7	60
m6g.12xlarge	7	60
m6g.16xlarge	14	120
m6gd.metal	14	120
m6gd.medium	1	4
m6gd.large	2	10
m6gd.xlarge	3	20
m6gd.2xlarge	3	40
m6gd.4xlarge	7	60
m6gd.8xlarge	7	60
m6gd.12xlarge	7	60
m6gd.16xlarge	14	120
m6gd.metal	14	120

p3 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
p3.2xlarge	3	40
p3.8xlarge	7	60
p3.16xlarge	7	120

r5 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
r5.large	2	10
r5.xlarge	3	20
r5.2xlarge	3	40

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
r5.4xlarge	7	60
r5.12xlarge	7	60
r5.16xlarge	14	120
r5.24xlarge	14	120
r5a.large	2	10
r5a.xlarge	3	20
r5a.2xlarge	3	40
r5a.4xlarge	7	60
r5a.8xlarge	7	60
r5a.12xlarge	7	60
r5a.16xlarge	14	120
r5a.24xlarge	14	120
r5ad.large	2	10
r5ad.xlarge	3	20
r5ad.2xlarge	3	40
r5ad.4xlarge	7	60
r5ad.8xlarge	7	60
r5ad.12xlarge	7	60
r5ad.16xlarge	14	120
r5ad.24xlarge	14	120
r5d.large	2	10
r5d.xlarge	3	20
r5d.2xlarge	3	40
r5d.4xlarge	7	60
r5d.8xlarge	7	60
r5d.12xlarge	7	60
r5d.16xlarge	14	120
r5d.24xlarge	14	120

r6 instance family

Instance type	Task limit without ENI trunking enabled	Task limit with ENI trunking enabled
r6g.medium	1	4
r6g.large	2	10
r6g.xlarge	3	20
r6g.2xlarge	3	40
r6g.4xlarge	7	60
r6g.8xlarge	7	60
r6g.12xlarge	7	60
r6g.16xlarge	14	120
r6g.metal	14	120
r6gd.medium	1	4
r6gd.large	2	10
r6gd.xlarge	3	20
r6gd.2xlarge	3	40
r6gd.4xlarge	7	60
r6gd.8xlarge	7	60
r6gd.12xlarge	7	60
r6gd.16xlarge	14	120
r6gd.metal	14	120

Container Instance Memory Management

When the Amazon ECS container agent registers a container instance into a cluster, the agent must determine how much memory the container instance has available to reserve for your tasks. Because of platform memory overhead and memory occupied by the system kernel, this number is different than the installed memory amount that is advertised for Amazon EC2 instances. For example, an m4.1large instance has 8 GiB of installed memory. However, this does not always translate to exactly 8192 MiB of memory available for tasks when the container instance registers.

If you specify 8192 MiB for the task, and none of your container instances have 8192 MiB or greater of memory available to satisfy this requirement, then the task cannot be placed in your cluster.

You should also reserve some memory for the Amazon ECS container agent and other critical system processes on your container instances, so that your task's containers do not contend for the same memory and possibly trigger a system failure. For more information, see [Reserving System Memory \(p. 383\)](#).

The Amazon ECS container agent uses the Docker `ReadMemInfo()` function to query the total memory available to the operating system. Both Linux and Windows provide command line utilities to determine the total memory.

Example - Determine Linux total memory

The `free` command returns the total memory that is recognized by the operating system.

```
$ free -b
```

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

	total	used	free	shared	buffers	cached
Mem:	8373026816	348180480	8024846336	90112	25534464	205418496
-/+ buffers/cache:	117227520	8255799296				

This instance has 8373026816 bytes of total memory, which translates to 7985 MiB available for tasks.

Example - Determine Windows total memory

The `wmic` command returns the total memory that is recognized by the operating system.

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

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

TotalPhysicalMemory
8589524992

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

Reserving System Memory

If you occupy all of the memory on a container instance with your tasks, then it is possible that your tasks will contend with critical system processes for memory and possibly trigger a system failure. The Amazon ECS container agent provides a configuration variable called `ECS_RESERVED_MEMORY`, which you can use to remove a specified number of MiB of memory from the pool that is allocated to your tasks. This effectively reserves that memory for critical system processes.

For example, if you specify `ECS_RESERVED_MEMORY=256` in your container agent configuration file, then the agent registers the total memory minus 256 MiB for that instance, and 256 MiB of memory could not be allocated by ECS tasks. For more information about agent configuration variables and how to set them, see [Amazon ECS container agent configuration \(p. 454\)](#) and [Bootstrapping container instances with Amazon EC2 user data \(p. 368\)](#).

Viewing Container Instance Memory

You can view how much memory a container instance registers with in the Amazon ECS console (or with the `DescribeContainerInstances` API operation). If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, you can observe the memory available for that container instance and then assign your tasks that much memory.

To view container instance memory

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.

2. Choose the cluster that hosts your container instances to view.
3. Choose **ECS Instances**, and select a container instance from the **Container Instance** column to view.
4. The **Resources** section shows the registered and available memory for the container instance.

Resources

Resources	Registered	Available
CPU	2048	2048
Memory	7953	7953
Ports	<i>5 ports</i>	

The **Registered** memory value is what the container instance registered with Amazon ECS when it was first launched, and the **Available** memory value is what has not already been allocated to tasks.

Connect to your container instance

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

- Your container instances need external network access to connect using SSH. If your container instances are running in a private VPC, they need an SSH bastion instance to provide this access. For more information, see the [Securely connect to Linux instances running in a private Amazon VPC](#) blog post.
- Your container instances must have been launched with a valid Amazon EC2 key pair. Amazon ECS container instances have no password, and you use a key pair to log in using SSH. If you did not specify a key pair when you launched your instance, there is no way to connect to the instance. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).
- SSH uses port 22 for communication. Port 22 must be open in your container instance security group for you to connect to your instance using SSH.

Note

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

To connect to your container instance

1. Find the public IP or DNS address for your container instance.
 - a. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
 - b. Select the cluster that hosts your container instance.

- c. On the **Cluster** page, choose **ECS Instances**.
 - d. On the **Container Instance** column, select the container instance to connect to.
 - e. On the **Container Instance** page, record the **Public IP** or **Public DNS** for your instance.
2. Find the default username for your container instance AMI. The user name for instances launched with an Amazon ECS-optimized AMI is `ec2-user`. For Ubuntu AMIs, the default user name is `ubuntu`.
 3. If you are using a macOS or Linux computer, connect to your instance with the following command, substituting the path to your private key and the public address for your instance:

```
$ ssh -i /path/to/my-key-pair.pem ec2-user@ec2-198-51-100-1.compute-1.amazonaws.com
```

For more information about using a Windows computer, see [Connecting to Your Linux Instance from Windows Using PuTTY](#) in the *Amazon EC2 User Guide for Linux Instances*.

Important

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

Manage container instances remotely using AWS Systems Manager

You can use the Run Command capability in AWS Systems Manager (Systems Manager) to securely and remotely manage the configuration of your Amazon ECS container instances. Run Command provides a simple way to perform common administrative tasks without logging on locally to the instance. You can manage configuration changes across your clusters by simultaneously executing commands on multiple container instances. Run Command reports the status and results of each command.

Here are some examples of the types of tasks you can perform with Run Command:

- Install or uninstall packages.
- Perform security updates.
- Clean up Docker images.
- Stop or start services.
- View system resources.
- View log files.
- Perform file operations.

For more information about Run Command, see [AWS Systems Manager Run Command](#) in the *AWS Systems Manager User Guide*.

Topics

- [Run Command IAM policy \(p. 385\)](#)
- [Using Run Command \(p. 386\)](#)

Run Command IAM policy

Before you can send commands to your container instances with Run Command, you must attach an IAM policy that allows `ecsInstanceRole` to have access to the Systems Manager APIs. The following

procedure describes how to attach the Systems Manager managed policies to your container instance role so that instances launched with this role can use Run Command.

To attach the Systems Manager policies to your `ecsInstanceRole`

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
 2. In the navigation pane, choose **Roles**.
 3. Choose `ecsInstanceRole`. If the role does not exist, follow the procedures in [Amazon ECS container instance IAM role \(p. 695\)](#) to create the role.
 4. Choose the **Permissions** tab.
 5. Choose **Attach policies**.
 6. To narrow the available policies to attach, for **Filter**, type `SSM`.
 7. In the list of policies, select the box next **AmazonSSMManagedInstanceCore**. This policy enables you to provide the minimum permissions that are necessary to use Systems Manager.
- For information about other policies you can provide for Systems Manager operations, see [Create an IAM Instance Profile for Systems Manager](#) in the *AWS Systems Manager User Guide*.
8. Choose **Attach Policy**.

Using Run Command

After you attach Systems Manager managed policies to your `ecsInstanceRole` and verify that AWS Systems Manager Agent (SSM Agent) is installed on your container instances, you can start using Run Command to send commands to your container instances. For information about running commands and shell scripts on your instances and viewing the resulting output, see [Running Commands Using Systems Manager Run Command](#) and [Run Command Walkthroughs](#) in the *AWS Systems Manager User Guide*.

Example: To update container instance software with Run Command

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

1. [Attach Systems Manager managed policies to your `ecsInstanceRole`. \(p. 385\)](#)
2. Verify that SSM Agent is installed on your container instances. For more information, see [Manually install SSM Agent on EC2 instances for Linux](#).
3. Open the Systems Manager console at <https://console.aws.amazon.com/systems-manager>.
4. In the left navigation pane, choose **Run Command**, and then choose **Run command**.
5. For **Command document**, choose `AWS-RunShellScript`.
6. In the **Commands** section, enter the command or commands to send to your container instances. In this example, the following command updates the instance software:

```
$ yum update -y
```

7. In the **Target instances** section, select the boxes next to the container instances where you want to run the update command.
8. Choose **Run** to send the command to the specified instances.
9. (Optional) Choose the refresh icon to monitor the command status.
10. (Optional) In **Targets and output**, choose the button next to the instance ID, and then choose **View output**.

Windows instances

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

Note

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

Topics

- [Amazon ECS-optimized AMI \(p. 387\)](#)
- [Launching an Amazon ECS Windows container instance \(p. 408\)](#)
- [Bootstrapping Windows container instances with Amazon EC2 user data \(p. 411\)](#)
- [Connect to your container Windows instance \(p. 413\)](#)

Amazon ECS-optimized AMI

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

The Amazon ECS-optimized AMI metadata, including the AMI name, Amazon ECS container agent version, and ECS runtime version which includes the Docker version, for each variant can be retrieved programmatically. For more information, see [the section called “Retrieving Amazon ECS-Optimized AMI metadata” \(p. 395\)](#).

Amazon ECS-optimized AMI variants

The following Windows Server variants of the Amazon ECS-optimized AMI are available for your Amazon EC2 instances. For more information, see [Amazon EC2 Windows containers \(p. 842\)](#).

- **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 2004 Core AMI**

Important

On December 14, 2021, the Amazon ECS-optimized Windows Server 2004 Core AMI reaches its end of support date. After this date, no new versions of this AMI will be released. For more information, see [Windows Server release information](#).

- **Amazon ECS-optimized Windows Server 20H2 Core AMI**
- **Amazon ECS-optimized Windows Server 2016 Full AMI**

Windows Server 2022, Windows Server 2019, and Windows Server 2016 are Long-Term Servicing Channel (LTSC) releases. Windows Server 2004 and Windows Server 20H2 are Semi-Annual Channel (SAC) releases. For more information, see [Windows Server release information](#).

The following are the details for retrieving the AMI IDs for each of the Windows variants of the Amazon ECS-optimized AMI. You can subscribe to the Windows AMI Amazon SNS topics to be notified when

a new AMI is released or an AMI version is marked private. For more information, see [Subscribing to Amazon ECS-optimized AMI update notifications \(p. 398\)](#).

Important

To ensure that customers have the latest security updates by default, Amazon ECS maintains at least the last three Windows Amazon ECS-optimized AMIs. After releasing new Windows Amazon ECS-optimized AMIs, Amazon ECS makes the Windows Amazon ECS-optimized AMIs that are older private. If there is a private AMI that you need access to, let us know by filing a ticket with Cloud Support.

Windows Server 2022 Full

The current Amazon ECS-optimized Windows Server 2022 Full AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

The following table provides a link to retrieve the current Amazon ECS-optimized Windows Server 2022 Full AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID

Region Name	Region	AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

For a full list of current and previous versions of the Windows Server 2022 Full and their corresponding versions of the Amazon ECS container agent and Docker see [Windows Amazon ECS-optimized AMIs versions \(p. 398\)](#).

Windows Server 2022 Core

The current Amazon ECS-optimized Windows Server 2022 Core AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

The following table provides a link to retrieve the current Amazon ECS-optimized Windows Server 2022 Core AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID

Region Name	Region	AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

For a full list of current and previous versions of the Windows Server 2022 Core and their corresponding versions of the Amazon ECS container agent and Docker see [Windows Amazon ECS-optimized AMIs versions \(p. 398\)](#).

Windows Server 2019 Full

The current Amazon ECS-optimized Windows Server 2019 Full AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

The following table provides a link to retrieve the current Amazon ECS-optimized Windows Server 2019 Full AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID

Region Name	Region	AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

For a full list of current and previous versions of the Windows Server 2019 Full and their corresponding versions of the Amazon ECS container agent and Docker see [Windows Amazon ECS-optimized AMIs versions \(p. 398\)](#).

Windows Server 2019 Core

The current Amazon ECS-optimized Windows Server 2019 Core AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

The following table provides a link to retrieve the current Amazon ECS-optimized Windows Server 2019 Core AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID

Region Name	Region	AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

For a full list of current and previous versions of the Windows Server 2019 Core and their corresponding versions of the Amazon ECS container agent and Docker see [Windows Amazon ECS-optimized AMIs versions \(p. 398\)](#).

Windows Server 2004 Core

The current Amazon ECS-optimized Windows Server 2004 Core AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2004-English-Core-ECS_Optimized
```

The following table provides a link to retrieve the current Amazon ECS-optimized Windows Server 2004 Core AMI IDs by Region.

Region Name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID

Region Name	Region	AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

For a full list of current and previous versions of the Amazon ECS-optimized Windows Server 2004 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker see [Windows Amazon ECS-optimized AMIs versions \(p. 398\)](#).

Important

On December 14, 2021, the Amazon ECS-optimized Windows Server 2004 Core AMI reaches its end of support date. After this date, no new versions of this AMI will be released. For more information, see [Windows Server release information](#).

Windows Server 20H2 Core

The current Amazon ECS-optimized Windows Server 20H2 Core AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-20H2-English-Core-ECS_Optimized
```

The following table provides a link to retrieve the current Amazon ECS-optimized Windows Server 20H2 Core AMI IDs by Region.

Region name	Region	AMI ID
US East (Ohio)	us-east-2	View AMI ID
US East (N. Virginia)	us-east-1	View AMI ID
US West (N. California)	us-west-1	View AMI ID
US West (Oregon)	us-west-2	View AMI ID
Asia Pacific (Hong Kong)	ap-east-1	View AMI ID
Asia Pacific (Tokyo)	ap-northeast-1	View AMI ID
Asia Pacific (Seoul)	ap-northeast-2	View AMI ID
Asia Pacific (Mumbai)	ap-south-1	View AMI ID
Asia Pacific (Singapore)	ap-southeast-1	View AMI ID
Asia Pacific (Sydney)	ap-southeast-2	View AMI ID
Canada (Central)	ca-central-1	View AMI ID

Region name	Region	AMI ID
Europe (Frankfurt)	eu-central-1	View AMI ID
Europe (Stockholm)	eu-north-1	View AMI ID
Europe (Ireland)	eu-west-1	View AMI ID
Europe (London)	eu-west-2	View AMI ID
Europe (Paris)	eu-west-3	View AMI ID
Middle East (Bahrain)	me-south-1	View AMI ID
South America (São Paulo)	sa-east-1	View AMI ID
AWS GovCloud (US-East)	us-gov-east-1	View AMI ID
AWS GovCloud (US-West)	us-gov-west-1	View AMI ID
China (Beijing)	cn-north-1	View AMI ID
China (Ningxia)	cn-northwest-1	View AMI ID

For a full list of current and previous versions of the Windows Server 20H2 Core and their corresponding versions of the Amazon ECS container agent and Docker see [Windows Amazon ECS-optimized AMIs versions \(p. 398\)](#).

Windows Server 2016 Full

The current Amazon ECS-optimized Windows Server 2016 Full AMI can be retrieved using the AWS CLI with the following command:

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

The following table provides a link to retrieve the current Amazon ECS-optimized Windows Server 2016 Full AMI IDs by Region.

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

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

For a full list of current and previous versions of the Amazon ECS-optimized Windows Server 2016 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker see [Windows Amazon ECS-optimized AMIs versions \(p. 398\)](#).

Retrieving Amazon ECS-Optimized AMI metadata

The AMI ID, image name, operating system, container agent version, and runtime version for each variant of the Amazon ECS-optimized AMIs can be programmatically retrieved by querying the Systems Manager Parameter Store API. For more information about the Systems Manager Parameter Store API, see [GetParameters](#) and [GetParametersByPath](#).

Note

Your user account must have the following IAM permissions to retrieve the Amazon ECS-optimized AMI metadata. These permissions have been added to the `AmazonECS_FullAccess` IAM policy.

- `ssm:GetParameters`
- `ssm:GetParameter`
- `ssm:GetParametersByPath`

Systems Manager Parameter Store parameter format

The following is the format of the parameter name for each Amazon ECS-optimized AMI variant.

- Windows Server 2022 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

- Windows Server 2022 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

- Windows Server 2019 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

- Windows Server 2019 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

- Windows Server 2004 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2004-English-Core-ECS_Optimized
```

Important

On December 14, 2021, the Amazon ECS-optimized Windows Server 2004 Core AMI reaches its end of support date. After this date, no new versions of this AMI will be released. For more information, see [Windows Server release information](#).

- Windows Server 20H2 Core AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-20H2-English-Core-ECS_Optimized
```

- Windows Server 2016 Full AMI metadata:

```
/aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

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

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

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

```
{  
    "schema_version": 1,  
    "image_name": "amzn2-ami-ecs-hvm-2.0.20181017-x86_64-ebs",  
    "image_id": "ami-04a4fb062c609f55b",  
    "os": "Amazon Linux 2",  
    "ecs_runtime_version": "Docker version 18.06.1-ce",  
    "ecs_agent_version": "1.21.0"  
}
```

Each of the fields in the output above are available to be queried as sub-parameters. Construct the parameter path for a sub-parameter by appending the sub-parameter name to the path for the selected AMI. The following sub-parameters are available:

- `schema_version`
- `image_id`
- `image_name`
- `os`
- `ecs_agent_version`
- `ecs_runtime_version`

The following parameter name format retrieves the image ID of the latest stable Amazon ECS-optimized Amazon Linux 2 AMI by using the sub-parameter `image_id`.

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

The following parameter name format retrieves the metadata of a specific Amazon ECS-optimized AMI version by specifying the AMI name.

- Amazon ECS-optimized Amazon Linux 2 AMI metadata:

```
/aws/service/ecs/optimized-ami/amazon-linux-2/amzn2-ami-ecs-hvm-2.0.20181112-x86_64-ebs
```

Note

All versions of the Amazon ECS-optimized Amazon Linux 2 AMI are available for retrieval. Only Amazon ECS-optimized AMI versions `amzn-ami-2017.09.1-amazon-ecs-optimized` (Linux) and later can be retrieved. For more information, see [Amazon ECS-optimized AMI versions \(p. 342\)](#).

Examples

The following examples show ways in which you can retrieve the metadata for each Amazon ECS-optimized AMI variant.

Retrieving the metadata of the latest stable Amazon ECS-optimized AMI

You can retrieve the latest stable Amazon ECS-optimized AMI using the AWS CLI with the following AWS CLI commands.

- **For the Amazon ECS-optimized Windows Server 2022 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2022 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2019 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2019 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2004 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2004-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 20H2 Core AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-20H2-English-Core-ECS_Optimized --region us-east-1
```

- **For the Amazon ECS-optimized Windows Server 2016 Full AMI:**

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized --region us-east-1
```

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

You can reference the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template by referencing the Systems Manager parameter store name.

```
Parameters:  
LatestECSOptimizedAMI:  
  Description: AMI ID  
  Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>  
  Default: /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized/image_id
```

Subscribing to Amazon ECS-optimized AMI update notifications

AWS provides two Amazon SNS topic ARNs for notifications related to the Windows Server Amazon Machine Image (AMIs). One topic sends update notifications when new Windows Server AMIs are released. The other topic sends notifications when previously released Windows Server AMIs are made private. While these topics are not specific to the Amazon ECS-optimized Windows AMIs, because the Amazon ECS-optimized Windows AMIs follow the same release schedule, you can use these notifications for an indication for when new Amazon ECS-optimized Windows AMIs are updated. For more information on subscribing to Windows AMI notifications, see [Subscribing to Windows AMI notifications](#) in the *Amazon EC2 User Guide for Windows Instances*.

Note

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

Amazon ECS-optimized AMI versions

This topic lists the current and previous versions of the Amazon ECS-optimized AMIs and their corresponding versions of the Amazon ECS container agent, Docker, and the `ecs-init` package.

The Amazon ECS-optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see [the section called “Retrieving Amazon ECS-Optimized AMI metadata” \(p. 395\)](#).

Windows Amazon ECS-optimized AMIs versions

The following tabs display a list of Windows Amazon ECS-optimized AMIs versions.

Note

For details on referencing the Systems Manager Parameter Store parameter in an AWS CloudFormation template, see [Using the latest recommended Amazon ECS-optimized AMI in an AWS CloudFormation template \(p. 342\)](#).

Important

To ensure that customers have the latest security updates by default, Amazon ECS maintains at least the last three Windows Amazon ECS-optimized AMIs. After releasing new Windows Amazon ECS-optimized AMIs, Amazon ECS makes the Windows Amazon ECS-optimized AMIs that are older private. If there is a private AMI that you need access to, let us know by filing a ticket with Cloud Support.

Windows Server 2022 Full AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2022 Full AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2022 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Full-ECS_Optimized-2021.009.23	1.55.3	20.10.7	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2022 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Full-ECS_Optimized
```

Windows Server 2022Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2022 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2022 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2022-English-Core-ECS_Optimized-2021.009.23	1.55.3	20.10.7	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2022 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-ECS_Optimized
```

Windows Server 2019 Full AMI versions

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

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-English-Full-ECS_Optimized-2021.009.23	1.55.3	20.10.7	Public

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019- English-Full-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Public
Windows_Server-2019- English-Full-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Public
Windows_Server-2019- English-Full-ECS_Optimized-2021.07.08	1.54.0	20.10.5	Public
Windows_Server-2019- English-Full-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Public
Windows_Server-2019- English-Full-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Public
Windows_Server-2019- English-Full-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Public
Windows_Server-2019- English-Full-ECS_Optimized-2021.03.11	1.50.2	19.03.14	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.02.10	1.50.0	19.03.14	Private
Windows_Server-2019- English-Full-ECS_Optimized-2021.01.13	1.49.0	19.03.14	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.11.18	1.48.0	19.03.13	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.11.06	1.47.0	19.03.11	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.10.14	1.45.0	19.03.11	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.08.12	1.43.0	19.03.11	Private
Windows_Server-2019- English-Full-ECS_Optimized-2020.07.15	1.41.1	19.03.5	Private

Amazon ECS-optimized Windows Server 2019 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-1.40.0-English-Full-ECS_Optimized-2020.06.11	1.40.0	19.03.5	Private
Windows_Server-2019-1.39.0-English-Full-ECS_Optimized-2020.05.14	1.39.0	19.03.5	Private
Windows_Server-2019-1.35.0-English-Full-ECS_Optimized-2020.01.15	1.35.0	19.03.5	Private
Windows_Server-2019-1.34.0-English-Full-ECS_Optimized-2019.12.16	1.34.0	19.03.5	Private
Windows_Server-2019-1.34.0-English-Full-ECS_Optimized-2019.11.25	1.34.0	19.03.4	Private
Windows_Server-2019-1.32.1-English-Full-ECS_Optimized-2019.11.13	1.32.1	19.03.4	Private
Windows_Server-2019-1.32.0-English-Full-ECS_Optimized-2019.10.09	1.32.0	19.03.2	Private
Windows_Server-2019-1.30.0-English-Full-ECS_Optimized-2019.09.11	1.30.0	19.03.1	Private
Windows_Server-2019-1.29.1-English-Full-ECS_Optimized-2019.08.16	1.29.1	19.03.1	Private
Windows_Server-2019-1.29.0-English-Full-ECS_Optimized-2019.07.19	1.29.0	18.09.8	Private
Windows_Server-2019-1.27.0-English-Full-ECS_Optimized-2019.05.10	1.27.0	18.09.4	Private

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2019 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Full-ECS_Optimized
```

Windows Server 2019 Core AMI versions

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

Amazon ECS-optimized Windows Server 2019 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-English-Core-ECS_Optimized-2021.09.23	1.55.3	20.10.7	Public
Windows_Server-2019-English-Core-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Public
Windows_Server-2019-English-Core-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Public
Windows_Server-2019-English-Core-ECS_Optimized-2021.07.08	1.54.0	20.10.6	Public
Windows_Server-2019-English-Core-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Public
Windows_Server-2019-English-Core-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Public
Windows_Server-2019-English-Core-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Public
Windows_Server-2019-English-Core-ECS_Optimized-2021.03.11	1.50.2	19.03.14	Private
Windows_Server-2019-English-Core-ECS_Optimized-2021.02.10	1.50.0	19.03.14	Private
Windows_Server-2019-English-Core-ECS_Optimized-2021.01.13	1.49.0	19.03.14	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.11.18	1.48.0	19.03.13	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.11.06	1.47.0	19.03.11	Private

Amazon ECS-optimized Windows Server 2019 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2019-English-Core-ECS_Optimized-2020.10.14	1.45.0	19.03.11	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.09.09	1.44.3	19.03.11	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.08.12	1.43.0	19.03.11	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.07.15	1.41.1	19.03.5	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.06.11	1.40.0	19.03.5	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.05.14	1.39.0	19.03.5	Private
Windows_Server-2019-English-Core-ECS_Optimized-2020.01.15	1.35.0	19.03.5	Private
Windows_Server-2019-English-Core-ECS_Optimized-2019.12.16	1.34.0	19.03.5	Private
Windows_Server-2019-English-Core-ECS_Optimized-2019.11.25	1.34.0	19.03.4	Private
Windows_Server-2019-English-Core-ECS_Optimized-2019.11.13	1.32.1	19.03.4	Private
Windows_Server-2019-English-Core-ECS_Optimized-2019.10.09	1.32.0	19.03.2	Private

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2019 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-ECS_Optimized
```

Windows Server 2016 Full AMI versions

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

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016-English-Full-ECS_Optimized-2021.09.23	1.55.3	20.10.7	Public
Windows_Server-2016-English-Full-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Public
Windows_Server-2016-English-Full-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Public
Windows_Server-2016-English-Full-ECS_Optimized-2021.07.08	1.54.0	20.10.5	Public
Windows_Server-2016-English-Full-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Public
Windows_Server-2016-English-Full-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Public
Windows_Server-2016-English-Full-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Public
Windows_Server-2016-English-Full-ECS_Optimized-2021.03.11	1.50.2	19.03.14	Private
Windows_Server-2016-English-Full-ECS_Optimized-2021.02.10	1.50.0	19.03.14	Private
Windows_Server-2016-English-Full-ECS_Optimized-2021.01.13	1.49.0	19.03.14	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.11.18	1.48.0	19.03.13	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.11.06	1.47.0	19.03.11	Private

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016-English-Full-ECS_Optimized-2020.10.14	1.45.0	19.03.12	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.09.09	1.44.3	19.03.11	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.08.12	1.43.0	19.03.11	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.07.15	1.41.1	19.03.5	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.06.11	1.40.0	19.03.5	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.05.14	1.39.0	19.03.5	Private
Windows_Server-2016-English-Full-ECS_Optimized-2020.01.15	1.35.0	19.03.5	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.12.16	1.34.0	19.03.5	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.11.25	1.34.0	19.03.4	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.11.13	1.32.1	19.03.4	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.10.09	1.32.0	19.03.2	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.09.11	1.30.0	19.03.1	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.08.16	1.29.1	19.03.1	Private
Windows_Server-2016-English-Full-ECS_Optimized-2019.07.19	1.29.0	18.09.8	Private

Amazon ECS-optimized Windows Server 2016 Full AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2016-English-Full-ECS_Optimized-2019.03.07	1.26.0	18.03.1	Private

Use the following AWS CLI Amazon ECS-optimized Windows Server 2016 Full AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2016-English-Full-ECS_Optimized
```

Windows Server 2004 Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 2004 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 2004 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2004-English-Core-ECS_Optimized-2021.09.23	1.55.3	20.10.7	Public
Windows_Server-2004-English-Core-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Public
Windows_Server-2004-English-Core-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Public
Windows_Server-2004-English-Core-ECS_Optimized-2021.07.08	1.54.0	20.10.5	Public
Windows_Server-2004-English-Core-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Public
Windows_Server-2004-English-Core-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Public
Windows_Server-2004-English-Core-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Public
Windows_Server-2004-English-Core-ECS_Optimized-2021.03.11	1.50.2	19.03.14	Private

Amazon ECS-optimized Windows Server 2004 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-2004-English-Core-ECS_Optimized-2021.02.10	1.50.0	19.03.14	Private
Windows_Server-2004-English-Core-ECS_Optimized-2021.01.13	1.49.0	19.03.14	Private
Windows_Server-2004-English-Core-ECS_Optimized-2020.11.18	1.48.0	19.03.13	Private
Windows_Server-2004-English-Core-ECS_Optimized-2020.11.06	1.47.0	19.03.11	Private
Windows_Server-2004-English-Core-ECS_Optimized-2020.10.14	1.45.0	19.03.12	Private
Windows_Server-2004-English-Core-ECS_Optimized-2020.09.09	1.44.3	19.03.11	Private
Windows_Server-2004-English-Core-ECS_Optimized-2020.08.12	1.43.0	19.03.11	Private

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 2004 Core AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-2004-English-Core-ECS_Optimized
```

Windows Server 20H2 Core AMI versions

The table below lists the current and previous versions of the Amazon ECS-optimized Windows Server 20H2 Core AMI and their corresponding versions of the Amazon ECS container agent and Docker.

Amazon ECS-optimized Windows Server 20H2 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-20H2-English-Core-ECS_Optimized-2021.08.12	1.55.0	20.10.6	Public
Windows_Server-20H2-English-Core-ECS_Optimized-2021.07.13	1.54.02	20.10.6	Public

Amazon ECS-optimized Windows Server 20H2 Core AMI	Amazon ECS container agent version	Docker version	Visibility
Windows_Server-20H2-English-Core-ECS_Optimized-2021.07.08	1.54.0	20.10.5	Public
Windows_Server-20H2-English-Core-ECS_Optimized-2021.06.11	1.53.0	20.10.5	Public
Windows_Server-20H2-English-Core-ECS_Optimized-2021.05.21	1.52.2	20.10.4	Public
Windows_Server-20H2-English-Core-ECS_Optimized-2021.04.14	1.51.0	20.10.0	Public

Use the following AWS CLI command to retrieve the current Amazon ECS-optimized Windows Server 20H2 Core AMI.

```
aws ssm get-parameters --names /aws/service/ami-windows-latest/Windows_Server-20H2-English-Core-ECS_Optimized
```

Launching an Amazon ECS Windows container instance

Your Amazon ECS container instances are created using the Amazon EC2 console. Before you begin, be sure that you've completed the steps in [Setting up with Amazon ECS \(p. 7\)](#).

To launch a container instance

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select the Region to use.
3. From the **EC2 Dashboard**, choose **Launch instance**.
4. On the **Choose an Amazon Machine Image (AMI)** page, complete the following steps:
 - a. Choose **AWS Marketplace**.
 - b. Choose an AMI for your container instance. You can search for one of the Amazon ECS-optimized AMIs, for example the **Windows_2019_Full_ECS_Optimized**. If you do not choose an Amazon ECS-optimized AMI, you must follow the procedures in [Installing the Amazon ECS container agent \(p. 431\)](#).

For more information about the latest Amazon ECS-optimized AMIs, see

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

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

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

- a. Set the **Number of instances** field depending on how many container instances you want to add to your cluster.
- b. (Optional) To use Spot Instances, for **Purchasing option**, select the check box next to **Request Spot Instances**. You also need to set the other fields related to Spot Instances. For more information, see [Spot Instance Requests](#).

Note

If you are using Spot Instances and see a `Not available` message, you may need to choose a different instance type.

- c. For **Network**, choose the VPC into which to launch your container instance.
- d. For **Subnet**, choose a subnet to use, or keep the default option to choose the default subnet in any Availability Zone.
- e. Set the **Auto-assign Public IP** field depending on whether you want your instance to be accessible from the public internet. If your instance should be accessible from the internet, verify that the **Auto-assign Public IP** field is set to **Enable**. If not, set this field to **Disable**.

Note

Container instances need access to communicate with the Amazon ECS service endpoint. This can be through an interface VPC endpoint or through your container instances having public IP addresses.

For more information about interface VPC endpoints, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\) \(p. 714\)](#).

If you do not have an interface VPC endpoint configured and your container instances do not have public IP addresses, then they must use network address translation (NAT) to provide this access. For more information, see [NAT gateways](#) in the *Amazon VPC User Guide* and [HTTP proxy configuration \(p. 504\)](#) in this guide. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters \(p. 727\)](#).

- f. Select your container instance IAM role. This is usually named `ecsInstanceRole`.

Important

If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent cannot connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

- g. Configure your Amazon ECS container instance with user data, such as the agent environment variables from [Amazon ECS container agent configuration \(p. 454\)](#). Amazon EC2 user data scripts are executed only one time, when the instance is first launched. The following are common examples of what user data is used for:
 - By default, your container instance launches into your default cluster. To launch into a non-default cluster, choose the **Advanced Details** list. Then, paste the following script into the **User data** field, replacing `your_cluster_name` with the name of your cluster.

The `EnableTaskIAMRole` turns on the Task IAM roles feature for the tasks.

In addition, the following options are available when you use the `awsvpc` network mode.

- `EnableTaskENI`: This flag turns on task networking and is required when you use the `awsvpc` network mode.
- `AwsvpcBlockIMDS`: This optional flag blocks IMDS access for the task containers running in the `awsvpc` network mode.
- `AwsvpcAdditionalLocalRoutes`: This optional flag allows you to have additional routes in the task namespace.

Replace `ip-address` with the IP Address for the additional routes, for example `172.31.42.23/32`.

```
<powershell>
```

```
Import-Module ECSTools
Initialize-ECSAgent -Cluster your_cluster_name -EnableTaskIAMRole -EnableTaskENI
-AwsVpcBlockIMDS -AwsVpcAdditionalLocalRoutes
'[{"ip-address"}]'
</powershell>
```

- h. Choose **Next: Add Storage**.
7. On the **Add Storage** page, configure the storage for your container instance.

You can optionally increase or decrease the volume sizes for your instance to meet your application needs.

When done configuring your volumes, choose **Next: Add Tags**.
8. On the **Add Tags** page, specify tags by providing key and value combinations for the container instance. Choose **Add another tag** to add more than one tag to your container instance. For more information resource tags, see [Resources and tags \(p. 604\)](#).
- Choose **Next: Configure Security Group** when you are done.
9. On the **Configure Security Group** page, use a security group to define firewall rules for your container instance. These rules specify which incoming network traffic is delivered to your container instance. All other traffic is ignored. Select or create a security group as follows, and then choose **Review and Launch**.
10. On the **Review Instance Launch** page, under **Security Groups**, you see that the wizard created and selected a security group for you. Instead, select the security group that you created in [Setting up with Amazon ECS \(p. 7\)](#) using the following steps:
 - a. Choose **Edit security groups**.
 - b. On the **Configure Security Group** page, select the **Select an existing security group** option.
 - c. Select the security group you created for your container instance from the list of existing security groups, and choose **Review and Launch**.
11. On the **Review Instance Launch** page, choose **Launch**.
12. In the **Select an existing key pair or create a new key pair** dialog box, choose **Choose an existing key pair**, then select the key pair that you created when getting set up.

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

13. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.
14. On the **Instances** screen, you can view the status of your instance. It takes a short time for an instance to launch. When you launch an instance, its initial state is pending. After the instance starts, its state changes to running, and it receives a public DNS name. If the **Public DNS** column is hidden, choose **Show/Hide, Public DNS**.

Using Spot Instances

A Spot Instance is an unused Amazon EC2 instance that is available for less than the On-Demand price. Because Spot Instances allow you to request unused EC2 instances at steep discounts, you can lower your Amazon EC2 costs significantly. The hourly price for a Spot Instance is called a Spot price. The Spot price of each instance type in each Availability Zone is set by Amazon EC2, and adjusted gradually based on the long-term supply of and demand for Spot Instances. For more information, see [Spot Instances](#) in the [Amazon EC2 User Guide for Windows Instances](#).

You can register Spot Instances to your Amazon ECS clusters. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

Spot Instance Draining

Amazon EC2 terminates, stops, or hibernates your Spot Instance when the Spot price exceeds the maximum price for your request or capacity is no longer available. Amazon EC2 provides a Spot Instance interruption notice, which gives the instance a two-minute warning before it is interrupted. If Amazon ECS Spot Instance draining is enabled on the instance, ECS receives the Spot Instance interruption notice and places the instance in `DRAINING` status.

Important

Amazon ECS monitors for the Spot Instance interruption notices that have the `terminate` and `stop` instance-actions. If you specified either the `hibernate` instance interruption behavior when requesting your Spot Instances or Spot Fleet, then Amazon ECS Spot Instance draining is not supported for those instances.

When a container instance is set to `DRAINING`, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Service tasks on the draining container instance that are in the `PENDING` state are stopped immediately. If there are container instances in the cluster that are available, replacement service tasks are started on them.

Spot Instance draining is disabled by default and must be manually enabled. To enable Spot Instance draining for a new container instance, when launching the container instance add the following script into the **User data** field, replacing `MyCluster` with the name of the cluster to register the container instance to.

```
<powershell>
Import-Module ECSTools
Initialize-ECSAgent -Cluster your_cluster_name -ECS_ENABLE_SPOT_INSTANCE_DRAINING
</powershell>
```

For more information, see [the section called “Launching a container instance” \(p. 408\)](#).

Bootstrapping Windows container instances with Amazon EC2 user data

When you launch an Amazon ECS container instance, you have the option of passing user data to the instance. The data can be used to perform common automated configuration tasks and even run scripts when the instance boots. For Amazon ECS, the most common use cases for user data are to pass configuration information to the Docker daemon and the Amazon ECS container agent.

You can pass multiple types of user data to Amazon EC2, including cloud booothooks, shell scripts, and `cloud-init` directives. For more information about these and other format types, see the [Cloud-Init documentation](#).

You can pass this user data when using the Amazon EC2 launch wizard. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

Topics

- [Default Windows user data \(p. 412\)](#)
- [Windows agent installation user data \(p. 412\)](#)
- [Windows IAM roles for tasks \(p. 413\)](#)

Default Windows user data

This example user data script shows the default user data that your Windows container instances receive if you use the [cluster creation wizard \(p. 176\)](#). The below script does the following:

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

In addition, the following options are available when you use the `awsvpc` network mode.

- `EnableTaskENI`: This flag turns on task networking and is required when you use the `awsvpc` network mode.
- `AwsVpcBlockIMDS`: This optional flag blocks IMDS access for the task containers running in `awsvpc` network mode.
- `AwsVpcAdditionalLocalRoutes`: This optional flag allows you to have additional routes.

Replace `ip-address` with the IP Address for the additional routes, for example `172.31.42.23/32`.

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

```
<powershell>
Initialize-ECSAgent -Cluster windows -EnableTaskIAMRole -LoggingDrivers '[{"json-
file", "awslogs"}]' -EnableTaskENI -AwsVpcBlockIMDS -AwsVpcAdditionalLocalRoutes
'[{"ip-address"}]'
</powershell>
```

Windows agent installation user data

This example user data script installs the Amazon ECS container agent on an instance launched with a **Windows_Server-2016-English-Full-Containers** AMI. It has been adapted from the agent installation instructions on the [Amazon ECS Container Agent GitHub repository](#) README page.

Note

This script is shared for example purposes. It is much easier to get started with Windows containers by using the Amazon ECS-optimized Windows Server AMI. For more information, see [Creating a cluster \(p. 176\)](#).

You can use this script for your own container instances (provided that they are launched with a version of the **Windows_Server-2016-English-Full-Containers** AMI). Be sure to replace the `windows` line to specify your own cluster name (if you are not using a cluster called windows).

```
<powershell>
# Set up directories the agent uses
New-Item -Type directory -Path ${env:ProgramFiles}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS -Force
New-Item -Type directory -Path ${env:ProgramData}\Amazon\ECS\data -Force
# Set up configuration
$ecsExeDir = "${env:ProgramFiles}\Amazon\ECS"
[Environment]::SetEnvironmentVariable("ECS_CLUSTER", "windows", "Machine")
[Environment]::SetEnvironmentVariable("ECS_LOGFILE", "${env:ProgramData}\Amazon\ECS\log
\ecs-agent.log", "Machine")
[Environment]::SetEnvironmentVariable("ECS_DATADIR", "${env:ProgramData}\Amazon\ECS\data",
"Machine")
```

```
# Download the agent
$agentVersion = "latest"
$agentZipUri = "https://s3.amazonaws.com/amazon-ecs-agent/ecs-agent-windows-
$agentVersion.zip"
$zipFile = "${env:TEMP}\ecs-agent.zip"
Invoke-RestMethod -OutFile $zipFile -Uri $agentZipUri
# Put the executables in the executable directory.
Expand-Archive -Path $zipFile -DestinationPath $ecsExeDir -Force
Set-Location ${ecsExeDir}
# Set $EnableTaskIAMRoles to $true to enable task IAM roles
# Note that enabling IAM roles will make port 80 unavailable for tasks.
[bool]$EnableTaskIAMRoles = $false
if ($EnableTaskIAMRoles) {
    $HostSetupScript = Invoke-WebRequest https://raw.githubusercontent.com/aws/amazon-ecs-
agent/master/misc/windows-deploy/hostsetup.ps1
    Invoke-Expression $($HostSetupScript.Content)
}
# Install the agent service
New-Service -Name "AmazonECS" ` 
    -BinaryPathName "$ecsExeDir\amazon-ecs-agent.exe -windows-service" ` 
    -DisplayName "Amazon ECS" ` 
    -Description "Amazon ECS service runs the Amazon ECS agent" ` 
    -DependsOn Docker ` 
    -StartupType Manual
sc.exe failure AmazonECS reset=300 actions=restart/5000/restart/30000/restart/60000
sc.exe failureflag AmazonECS 1
Start-Service AmazonECS
</powershell>
```

Windows IAM roles for tasks

See the following Windows examples regarding bootstrapping IAM task roles.

- [Windows IAM roles for tasks \(p. 846\)](#)
- [IAM roles for task container bootstrap script \(p. 847\)](#)

Connect to your container Windows instance

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

- Amazon EC2 instances created from most Windows AMIs allow you to connect using Remote Desktop. Remote Desktop uses the Remote Desktop Protocol (RDP) and allows you to connect to and use your instance in the same way you use a computer sitting in front of you. It is available on most editions of Windows and available for Mac OS.
- Your container instances must have been launched with a valid Amazon EC2 key pair. Amazon ECS container instances have no password, and you use a key pair to log in using SSH. If you did not specify a key pair when you launched your instance, there is no way to connect to the instance. For more information, see [the section called “Launching a container instance” \(p. 408\)](#)
- Ensure that the security group associated with your instance allows incoming RDP traffic (port 3389) from your IP address. The default security group does not allow incoming RDP traffic by default. For more information, see [Authorize inbound traffic for your Windows instances](#) in the *Amazon EC2 User Guide for Windows Instances*.

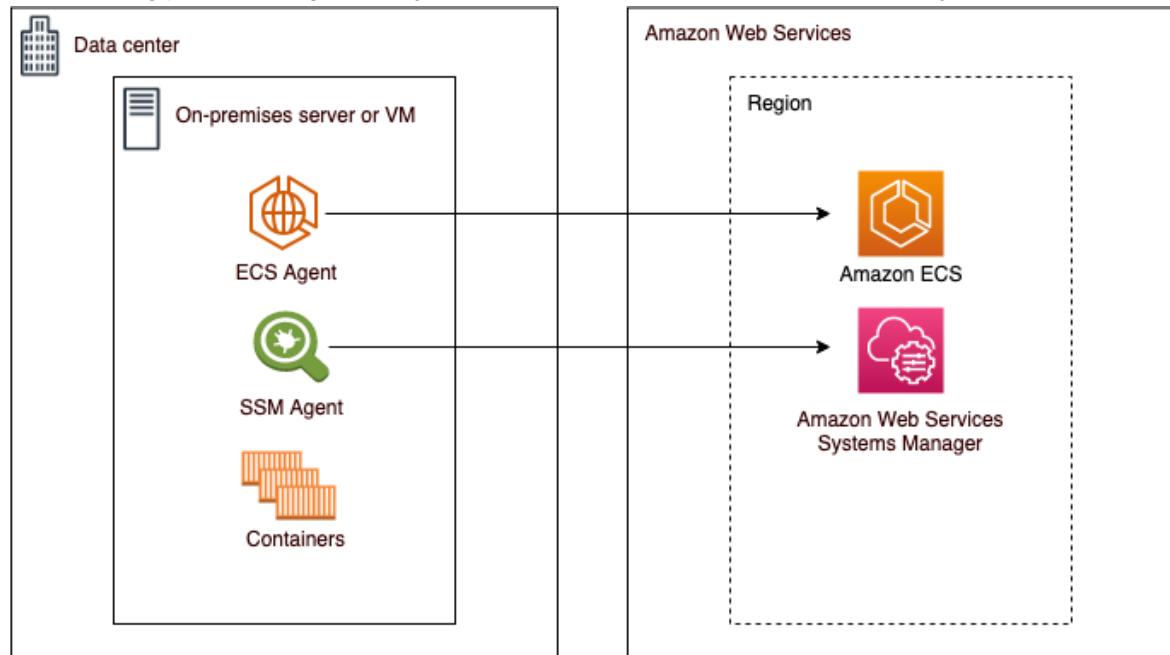
To connect to your container instance

1. Find the public IP or DNS address for your container instance.
 - a. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
 - b. Select the cluster that hosts your container instance.
 - c. On the **Cluster** page, choose **ECS Instances**.
 - d. On the **Container Instance** column, select the container instance to connect to.
 - e. On the **Container Instance** page, record the **Public IP** or **Public DNS** for your instance.
2. Find the default username for your container instance AMI.
3. You can connect to your instance by using RDP. For more information, see [Connect to your Windows instance using RDP](#) in the *Amazon EC2 User Guide for Windows Instances*.

External instances (Amazon ECS Anywhere)

Amazon ECS Anywhere provides support for registering an *external instance* such as an on-premises server or virtual machine (VM), to your Amazon ECS cluster. External instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, the lack of Elastic Load Balancing support makes running these workloads less efficient. Amazon ECS added a new **EXTERNAL** launch type that you can use to create services or run tasks on your external instances.

The following provides a high-level system architecture overview of Amazon ECS Anywhere.



Topics

- [Considerations \(p. 415\)](#)
- [IAM permissions for Amazon ECS Anywhere \(p. 416\)](#)
- [Registering an external instance to a cluster \(p. 418\)](#)
- [Deregistering an external instance \(p. 421\)](#)
- [Running workloads on external instances \(p. 422\)](#)

- [Updating the AWS Systems Manager Agent and Amazon ECS container agent on an external instance \(p. 423\)](#)

Considerations

Before you start using external instances, be aware of the following considerations:

- Currently, you can't register external instances with Amazon ECS Anywhere in the China (Beijing) and China (Ningxia) Regions.
- You can register an external instance to one cluster at a time. For instructions on how to register an external instance with a different cluster, see [Deregistering an external instance \(p. 421\)](#).
- Your external instances require an IAM role that allows them to communicate with AWS APIs. For more information, see [Required IAM permissions for external instances \(p. 416\)](#).
- Your external instances should not have a preconfigured instance credential chain defined locally as this will interfere with the registration script.
- To send container logs to CloudWatch Logs, make sure that you create and specify a task execution IAM role in your task definition. For more information, see [Conditional IAM permissions \(p. 418\)](#).
- When an external instance is registered to a cluster, the `ecs.capability.external` attribute is associated with the instance. This attribute identifies the instance as an external instance.
- You can add custom attributes to your external instances to use as a task placement constraint. For more information, see [Custom attributes \(p. 518\)](#).
- You can add resource tags to your external instance. For more information, see [Adding tags to an external container instance \(p. 609\)](#).
- The following are additional considerations that are specific to networking with your external instances. For more information, see [Networking with ECS Anywhere \(p. 416\)](#).
 - Service load balancing isn't supported.
 - Service discovery isn't supported.
 - Tasks that run on external instances must use the `bridge`, `host`, or `none` network modes. The `awsvpc` network mode isn't supported.
 - There are Amazon ECS service domains in each AWS Region. These service domains must be allowed to send traffic to your external instances.
 - The SSM Agent installed on your external instance maintains IAM credentials that are rotated every 30 minutes using a hardware fingerprint. If your external instance loses connection to AWS, the SSM Agent automatically refreshes the credentials after the connection is re-established. For more information, see [Validating on-premises servers and virtual machines using a hardware fingerprint in the AWS Systems Manager User Guide](#).
- The `UpdateContainerAgent` API isn't supported. For instructions on how to update the SSM Agent or the Amazon ECS agent on your external instances, see [Updating the AWS Systems Manager Agent and Amazon ECS container agent on an external instance \(p. 423\)](#).
- Amazon ECS capacity providers aren't supported. To create a service or run a standalone task on your external instances, use the `EXTERNAL` launch type.
- ECS Exec isn't supported.
- SELinux isn't supported.
- Using Amazon EFS volumes or specifying an `EFSVolumeConfiguration` isn't supported.
- Integration with App Mesh isn't supported.

Supported operating systems and system architectures

The following is the list of supported operating systems and system architectures.

- CentOS 7, CentOS 8
- RHEL 7 — Neither Docker or RHEL's open package repositories support installing Docker natively on RHEL. You must ensure that Docker is installed before you run the install script that's described in this document.
- Fedora 32, Fedora 33 — Fedora 32 and Fedora 33 default to using `cgroups.v2`, which isn't supported by Amazon ECS. As a result, the server's default grub configuration must be changed and the server rebooted. For instructions, see [Changing cgroup version](#) in the Docker documentation.
- openSUSE Tumbleweed
- Ubuntu 18, Ubuntu 20
- Debian 9, Debian 10
- SUSE Enterprise Server 15
- The `x86_64` and `ARM64` CPU architectures are supported.

Networking with ECS Anywhere

Amazon ECS external instances are optimized for running applications that generate outbound traffic or process data. If your application requires inbound traffic, such as a web service, the lack of Elastic Load Balancing support makes running these workloads less efficient because there isn't support for placing these workloads behind a load balancer.

The tasks you run on your external instances must use the `bridge`, `host`, or `none` network modes. The `awsvpc` network mode isn't supported. For more information about each network mode, see [Choosing a network mode](#) in the *Amazon ECS Best Practices Guide*.

The following domains are used for communication between the Amazon ECS service and the Amazon ECS agent that's installed on your external instance. Make sure that traffic is allowed and that DNS resolution works. For each endpoint, `region` represents the Region identifier for an AWS Region that's supported by Amazon ECS, such as `us-east-2` for the US East (Ohio) Region. The endpoints for all Regions that you use should be allowed. For the `ecs-a` and `ecs-t` endpoints, you should include an asterisk (for example, `ecs-a-*`).

- `ecs-a-*.region.amazonaws.com` — This endpoint is used when managing tasks.
- `ecs-t-*.region.amazonaws.com` — This endpoint is used to manage task and container metrics.
- `ecs.region.amazonaws.com` — This is the service endpoint for Amazon ECS.
- If your tasks require communication with any other AWS services, make sure that those service endpoints are allowed. Example applications include using Amazon ECR to pull container images or using CloudWatch for CloudWatch Logs. For more information, see [Service endpoints](#) in the *AWS General Reference*.

IAM permissions for Amazon ECS Anywhere

There are several required and conditional IAM permissions that apply to Amazon ECS Anywhere. The following sections describe the IAM permissions in more detail.

Required IAM permissions for external instances

When registering an on-premises server or virtual machine (VM) to your cluster, the server or VM requires an IAM role to communicate with AWS APIs. You only need to create this IAM role once for each AWS account. However, this IAM role must be associated with each server or VM that you register to a cluster. This role is the `ECSAnywhereRole`. You can create this role manually. Alternatively, Amazon ECS can create the role on your behalf when you register an external instance in the AWS Management Console.

AWS provides two managed IAM policies that can be used when creating the ECS Anywhere IAM role, the `AmazonSSMManagedInstanceCore` and `AmazonEC2ContainerServiceforEC2Role` policies. The `AmazonEC2ContainerServiceforEC2Role` policy includes permissions that likely provide more access than you need. Therefore, depending on your specific use case, we recommend that you create a custom policy adding only the permissions from that policy that you require in it. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

To create the ECS Anywhere 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 **Systems Manager**.
4. Choose the **Systems Manager** use case and then **Next: Permissions**.
5. In the **Attached permissions policy** section, search for and select the `AmazonSSMManagedInstanceCore` and `AmazonEC2ContainerServiceforEC2Role` policies and then choose **Next: Review**.

Important

The `AmazonEC2ContainerServiceforEC2Role` managed policy provides permissions that are needed for your on-premises server or VM. However, the `AmazonEC2ContainerServiceforEC2Role` managed policy might grant permissions that aren't needed for your use case. Review the permissions granted by this policy and see if your use case doesn't require all of the permissions. Then, depending on your situation, optionally create a custom policy and add only the permissions you require. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

6. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
7. For **Role name**, enter `ECSAnywhereRole` and optionally you can edit the description.
8. Review your role information and then choose **Create role**.
9. Perform a search for the `ECSAnywhereRole` and then select it to view the role details.
10. On the **Permissions** tab, choose **Attach policies**.
11. Search for the `AmazonSSMManagedInstanceCore` policy, select it, and then choose **Attach policy**.

To create the ECS Anywhere IAM role (AWS CLI)

1. Create a file named `ssm-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": [  
    {"Effect": "Allow",  
     "Principal": {"Service": [  
       "ssm.amazonaws.com"  
     ]},  
     "Action": "sts:AssumeRole"  
   }  
}
```

2. Create an IAM role named `ecsAnywhereRole` using the trust policy that's created in the previous step.

```
aws iam create-role \  
  --role-name ecsAnywhereRole \  
  --assume-role-policy-document file://ssm-trust-policy.json
```

3. Attach the AWS managed `AmazonSSMManagedInstanceCore` policy to the `ecsAnywhereRole` role. This policy provides the Systems Manager API permissions that are needed for your on-premises server or VM.

```
aws iam attach-role-policy \
--role-name ecsAnywhereRole \
--policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

4. Attach the AWS managed `AmazonEC2ContainerServiceforEC2Role` policy to the `ecsAnywhereRole` role.

```
aws iam attach-role-policy \
--role-name ecsAnywhereRole \
--policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEC2ContainerServiceforEC2Role
```

Important

The `AmazonEC2ContainerServiceforEC2Role` managed policy provides permissions that are needed for your on-premises server or VM. However, the `AmazonEC2ContainerServiceforEC2Role` managed policy might grant permissions that aren't needed for your use case. Review the permissions granted by this policy and see if your use case doesn't require all of the permissions. Then, depending on your situation, optionally create a custom policy and add only the permissions you require. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

Conditional IAM permissions

The task execution IAM role grants the Amazon ECS container agent permission to make AWS API calls on your behalf. When a task execution IAM role is used, it must be specified in your task definition. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#).

The task execution role is required if any of the following conditions apply:

- You're sending container logs to CloudWatch Logs using the `awslogs` log driver.
- Your task definition specifies a container image that's hosted in an Amazon ECR private repository. However, if the `ECSAnywhereRole` IAM role that's associated with your external instance also includes the permissions necessary to pull images from Amazon ECR then your task execution role doesn't need to include them.

Registering an external instance to a cluster

For each external instance you register with an Amazon ECS cluster, it must have the SSM Agent, the Amazon ECS container agent, and Docker installed. To register the external instance to an Amazon ECS cluster, it must first be registered as an AWS Systems Manager managed instance. You can create the installation script in a few clicks on the Amazon ECS console. The installation script includes an Systems Manager activation key and commands to install each of the required agents and Docker. The installation script must be run on your on-premises server or VM to complete the installation and registration steps.

Note

Before registering your external instance with the cluster, create the `/etc/ecs/ecs.config` file on your external instance and add any container agent configuration parameters that you want. You can't do this after registering the external instance to a cluster. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

To register an external instance (AWS Management 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, choose a cluster to register your external instance to.
5. Choose the **ECS Instances** tab, then choose **Register external instances**.
6. On the **Step 1: External instances activation details** page, complete the following steps.
 - a. For **Activation key duration (in days)**, enter the number of days that the activation key remains active for. After the number of days you entered pass, the key no longer works when registering an external instance.
 - b. For **Number of instances**, enter the number of external instances that you want to register to your cluster with the activation key.
 - c. For **Instance role**, choose the IAM role to associate with your external instances. If a role wasn't already created, choose **Create new role** to have Amazon ECS create a role on your behalf. For more information about what IAM permissions are required for your external instances, see [Required IAM permissions for external instances \(p. 416\)](#).
 - d. Choose **Next step**.
7. On the **Step 2: Register external instances** page, copy the registration command. This command should be run on each external instance you want to register to the cluster.

Important

The bash portion of the script must be run as root. If the command isn't run as root, an error is returned.

The AWS CLI can be used to create a Systems Manager activation before running the installation script to complete the external instance registration process.

To register an external instance (AWS CLI)

1. Create an Systems Manager activation pair. This is used for Systems Manager managed instance activation. The output includes an ActivationId and ActivationCode. You use these in a later step. Make sure that you specify the ECS Anywhere IAM role that you created. For more information, see [Required IAM permissions for external instances \(p. 416\)](#).

```
aws ssm create-activation --iam-role ecsAnywhereRole | tee ssm-activation.json
```

2. On your on-premises server or virtual machine (VM), download the installation script.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh"
```

3. (Optional) On your on-premises server or virtual machine (VM), use the following steps to verify the installation script using the script signature file.
 - a. Download and install GnuPG. For more information about GNUpg, see the [GnuPG website](#). For Linux systems, install gpg using the package manager on your flavor of Linux.
 - b. Retrieve the Amazon ECS PGP public key.

```
gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
```

3. (Optional) On your on-premises server or virtual machine (VM), use the following steps to verify the installation script using the script signature file.
 - c. Download the installation script signature. The signature is an ascii detached PGP signature stored in a file with the .asc extension.

```
curl --proto "https" -o "/tmp/ecs-anywhere-install.sh.asc" "https://amazon-ecs-agent.s3.amazonaws.com/ecs-anywhere-install-latest.sh.asc"
```

- d. Verify the installation script file using the key.

```
gpg --verify /tmp/ecs-anywhere-install.sh.asc /tmp/ecs-anywhere-install.sh
```

The following is the expected output.

```
gpg: Signature made Tue 25 May 2021 07:16:29 PM UTC
gpg:                               using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                               There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. On your on-premises server or virtual machine (VM), run the installation script. Specify the cluster name, Region, and the Systems Manager activation ID and activation code from the first step.

```
sudo bash /tmp/ecs-anywhere-install.sh \
--region $REGION \
--cluster $CLUSTER_NAME \
--activation-id $ACTIVATION_ID \
--activation-code $ACTIVATION_CODE
```

For an on-premises server or virtual machine (VM) that has the NVIDIA driver installed for GPU workloads, you must add the --enable-gpu flag to the installation script. When this flag is specified, the install script verifies that the NVIDIA driver is running and then adds the required configuration variables to run your Amazon ECS tasks. For more information about running GPU workloads and specifying GPU requirements in a task definition, see [Specifying GPUs in your task definition \(p. 252\)](#).

```
sudo bash /tmp/ecs-anywhere-install.sh \
--region $REGION \
--cluster $CLUSTER_NAME \
--activation-id $ACTIVATION_ID \
--activation-code $ACTIVATION_CODE \
--enable-gpu
```

Use the following steps to register an existing external instance with a different cluster.

To register an existing external instance with a different cluster

1. Stop the Amazon ECS container agent.

```
sudo systemctl stop ecs.service
```

2. Edit the /etc/ecs/ecs.config file and on the ECS_CLUSTER line, ensure the cluster name matches the name of the cluster to register the external instance with.
3. Remove the existing Amazon ECS agent data.

```
sudo rm /var/lib/ecs/data/agent.db
```

4. Start the Amazon ECS container agent.

```
sudo systemctl start ecs.service
```

Deregistering an external instance

We recommend that, after you finish using an external instance, you deregister the instance from both Amazon ECS and AWS Systems Manager. Following deregistration, the external instance is no longer able to accept new tasks.

If you have tasks that are running on the container instance when you deregister it, the tasks remain running until they stop through some other means. However, these tasks are no longer monitored or accounted for by Amazon ECS. If these tasks on your external instance are part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different instance, if possible.

To register an external instance to a new cluster, after the external instance has been deregistered from both Amazon ECS and Systems Manager, you can clean up the remaining AWS resources on the instance and register it with a new cluster.

To deregister an external instance (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, choose the Region where your external instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the external instance.
4. On the **Cluster : name** page, choose the **ECS Instances** tab.

Container Instance	EC2 Instance	Agent ...	Status	Availa...	Availa...
3de21d77-d1d7-4795-a3b3-ed6e65d7d353	i-501f2599	true	ACTIVE	2048	3955

5. Select the external instance ID to deregister. You're redirected to the container instance detail page.
6. On the **Container Instance : id** page, choose **Deregister**.
7. Review the deregistration message. Select **Deregister from AWS Systems Manager** to also deregister the external instance as an Systems Manager managed instance. Choose **Deregister**.

Note

You can deregister the external instance as an Systems Manager managed instance in the Systems Manager console. For instructions, see [Deregistering managed instances](#) in the [AWS Systems Manager User Guide](#).

8. If you want to clean up the remaining AWS resources from the external instance, follow these steps. The cleanup steps must be completed before you can register the external instance with a new cluster.

To clean up AWS resources on your on-premises server or VM

1. Make sure that the external instance is deregistered from both Amazon ECS and Systems Manager.
2. Stop the Amazon ECS container agent and the SSM Agent services on the instance.

```
sudo systemctl stop ecs amazon-ssm-agent
```

3. Remove the Amazon ECS and Systems Manager packages.

For CentOS 7, CentOS 8, and RHEL 7

```
sudo yum remove -y amazon-ecs-init amazon-ssm-agent
```

For SUSE Enterprise Server 15

```
sudo zypper remove -y amazon-ecs-init amazon-ssm-agent
```

For Debian and Ubuntu

```
sudo apt remove -y amazon-ecs-init amazon-ssm-agent
```

4. Remove the Amazon ECS and Systems Manager files.

```
sudo rm -rf /var/lib/ecs /etc/ecs /var/lib/amazon/ssm /var/log/ecs /var/log/amazon/ssm
```

Running workloads on external instances

After an external instance is registered to your cluster, you can run your containerized workloads. The first step is to register a task definition. Amazon ECS provides the `requiresCompatibilities` parameter to validate the task definition is compatible with your external instances. When you deploy your workload, use the `EXTERNAL` launch type when creating your service or running your standalone task.

Creating a task definition that is compatible with your external instances

When registering an Amazon ECS task definition, use the `requiresCompatibilities` parameter and specify `EXTERNAL` which validates that the task definition is compatible to use when running Amazon ECS workloads on your external instances. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

Important

If your tasks require a task execution IAM role, make sure that it's specified in the task definition. For more information, see [Conditional IAM permissions \(p. 418\)](#).

The following is an example task definition.

```
{
  "requiresCompatibilities": [
    "EXTERNAL"
  ],
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "public.ecr.aws/nginx/nginx:latest",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ]
    }
  ]
}
```

```
        "containerPort": 80,  
        "hostPort": 8080,  
        "protocol": "tcp"  
    }]  
],  
"networkMode": "bridge",  
"family": "nginx"  
}
```

Running a standalone task or creating a service

After you register your external instances to your cluster, grant the relevant IAM permissions, and register a valid task definition, you can start to run your workloads on Amazon ECS. When running your standalone tasks or creating a service, specify the `EXTERNAL` launch type, and the Amazon ECS scheduler places the tasks on your external instances.

For instructions on how to create services, see [Creating an Amazon ECS service \(p. 546\)](#).

For more information about running standalone tasks, see [Run a standalone task \(p. 510\)](#).

Updating the AWS Systems Manager Agent and Amazon ECS container agent on an external instance

Your on-premises server or VM must run both the AWS Systems Manager Agent (SSM Agent) and the Amazon ECS container agent when running Amazon ECS workloads. AWS releases new versions of these agents when any capabilities are added or updated. If your external instances are using an earlier version of either agent, you can update them using the following procedures.

Updating the SSM Agent on an external instance

AWS Systems Manager recommends that you automate the process of updating the SSM Agent on your instances. They provide several methods to automate updates. For more information, see [Automating updates to SSM Agent in the AWS Systems Manager User Guide](#).

Updating the Amazon ECS agent on an external instance

On your external instances, the Amazon ECS container agent is updated by upgrading the `ecs-init` package. Updating the Amazon ECS agent doesn't interrupt the running tasks or services. Amazon ECS provides the `ecs-init` package and signature file in an Amazon S3 bucket in each Region. Beginning with `ecs-init` version 1.52.1-1, Amazon ECS provides separate `ecs-init` packages for use depending on the operating system and system architecture your external instance uses.

Use the following table to determine the `ecs-init` package that you should download based on the operating system and system architecture your external instance uses.

Note

You can determine which operating system and system architecture that your external instance uses by using the following commands.

```
cat /etc/os-release  
uname -m
```

Operating systems (architecture)	ecs-init package
CentOS 7 (x86_64)	<code>amazon-ecs-init-latest.x86_64.rpm</code>

Operating systems (architecture)	ecs-init package
CentOS 8 (x86_64)	
SUSE Enterprise Server 15 (x86_64)	
RHEL 7 (x86_64)	
CentOS 7 (aarch64)	amazon-ecs-init-latest.aarch64.rpm
CentOS 8 (aarch64)	
RHEL 7 (aarch64)	
Debian 9 (x86_64)	amazon-ecs-init-latest.amd64.deb
Debian 10 (x86_64)	
Ubuntu 18 (x86_64)	
Ubuntu 20 (x86_64)	
Debian 9 (aarch64)	amazon-ecs-init-latest.arm64.deb
Debian 10 (aarch64)	
Ubuntu 18 (aarch64)	
Ubuntu 20 (aarch64)	

Follow these steps to update the Amazon ECS agent.

To update the Amazon ECS agent

1. Confirm the Amazon ECS agent version that you're running.

```
curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

2. Download the ecs-init package for your operating system and system architecture. Amazon ECS provides the ecs-init package file in an Amazon S3 bucket in each Region. Make sure that you replace the <region> identifier in the command with the Region name (for example, us-west-2) that you're geographically closest to.

amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb
```

3. (Optional) Verify the validity of the ecs-init package file using the PGP signature.
 - a. Download and install GnuPG. For more information about GNUpg, see the [GnuPG website](#). For Linux systems, install gpg using the package manager on your flavor of Linux.
 - b. Retrieve the Amazon ECS PGP public key.

```
gpg --keyserver hkp://keys.gnupg.net:80 --recv BCE9D9A42D51784F
```

- c. Download the ecs-init package signature. The signature is an ASCII detached PGP signature that's stored in a file with the .asc extension. Amazon ECS provides the signature file in an Amazon S3 bucket in each Region. Make sure that you replace the <region> identifier in the command with the Region name (for example, us-west-2) that you're geographically closest to.

amazon-ecs-init-latest.x86_64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.x86_64.rpm.asc
```

amazon-ecs-init-latest.aarch64.rpm

```
curl -o amazon-ecs-init.rpm.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.aarch64.rpm.asc
```

amazon-ecs-init-latest.amd64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.amd64.deb.asc
```

amazon-ecs-init-latest.arm64.deb

```
curl -o amazon-ecs-init.deb.asc https://s3.<region>.amazonaws.com/amazon-ecs-agent-<region>/amazon-ecs-init-latest.arm64.deb.asc
```

- d. Verify the ecs-init package file using the key.

For the rpm packages

```
gpg --verify amazon-ecs-init.rpm.asc ./amazon-ecs-init.rpm
```

For the deb packages

```
gpg --verify amazon-ecs-init.deb.asc ./amazon-ecs-init.deb
```

The following is the expected output.

```
gpg: Signature made Fri 14 May 2021 09:31:36 PM UTC
gpg:                               using RSA key 50DECCC4710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
```

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:                 There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

4. Install the `ecs-init` package.

For the rpm package on CentOS 7, CentOS 8, and RHEL 7

```
sudo yum install -y ./amazon-ecs-init.rpm
```

For the rpm package on SUSE Enterprise Server 15

```
sudo zypper install -y --allow-unsigned-rpm ./amazon-ecs-init.rpm
```

For the deb package

```
sudo dpkg -i ./amazon-ecs-init.deb
```

5. Restart the `ecs` service.

```
sudo systemctl restart ecs
```

6. Verify the Amazon ECS agent version was updated.

```
curl -s 127.0.0.1:51678/v1/metadata | python3 -mjson.tool
```

Monitoring your container instances

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

For information about sending container logs from your tasks to CloudWatch Logs, see [Using the awslogs log driver \(p. 283\)](#). For more information about CloudWatch Logs, see [Monitoring Log Files](#) in the [Amazon CloudWatch User Guide](#).

Topics

- [CloudWatch Logs IAM Policy \(p. 426\)](#)
- [Installing and configuring the CloudWatch agent \(p. 427\)](#)
- [Viewing CloudWatch Logs \(p. 427\)](#)

CloudWatch Logs IAM Policy

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

To create the `ECS-CloudWatchLogs` IAM policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.

3. Choose **Create policy, JSON**.
4. Enter the following policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:CreateLogGroup",  
                "logs:CreateLogStream",  
                "logs:PutLogEvents",  
                "logs:DescribeLogStreams"  
            ],  
            "Resource": [  
                "arn:aws:logs:*:*:  
            ]  
        }  
    ]  
}
```

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

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

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Choose **ecsInstanceRole**. If the role does not exist, follow the procedures in [Amazon ECS container instance IAM role \(p. 695\)](#) to create the role.
4. In the navigation pane, choose **Policies**.
5. Choose **ECS-CloudWatchLogs**.
6. Choose **Policy actions, Attach**.
7. To narrow the available policies to attach, for **Filter**, type **ecsInstance**.
8. Select the **ecsInstance** role and choose **Attach policy**.

Installing and configuring the CloudWatch agent

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

For more information, see [Download and configure the CloudWatch agent using the command line](#) in the [Amazon CloudWatch User Guide](#).

Viewing CloudWatch Logs

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

Note

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

To view your CloudWatch Logs data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. In the left navigation pane, choose **Logs, Log groups**.

Log Groups

Create Metric Filter **Actions**

Filter: Log Group Name Prefix

Log Groups
<input type="checkbox"/> /var/log/dmesg
<input type="checkbox"/> /var/log/docker
<input type="checkbox"/> /var/log/ecs/ecs-agent.log
<input type="checkbox"/> /var/log/ecs/ecs-init.log
<input type="checkbox"/> /var/log/messages

3. Choose a log group to view.
4. Choose a log stream to view. The streams are identified by the cluster name and container instance ID that sent the logs.

Filter: Search for events

Date/Time: 2016/02/11 20:31:02 UTC (GMT)

Event Data

- ▼ 2016-02-11T20:31:02Z [INFO] Starting Agent: Amazon ECS Agent - v1.7.1 (007985c)
- ▼ 2016-02-11T20:31:02Z [INFO] Loading configuration
- ▼ 2016-02-11T20:31:02Z [INFO] Checkpointing is enabled. Attempting to load state
- ▼ 2016-02-11T20:31:02Z [INFO] Loading state! module="statemanager"
- ▼ 2016-02-11T20:31:02Z [INFO] Detected Docker versions [1.17 1.18 1.19 1.20]
- ▼ 2016-02-11T20:31:02Z [INFO] Registering Instance with ECS
- ▼ 2016-02-11T20:31:02Z [INFO] Registered! module="api client"
- ▼ 2016-02-11T20:31:02Z [INFO] Registration completed successfully. I am running as 'arn:aws:ecs:us-east-1:0123456789:container-instance/07dfaf0a-eded-42c9-9cd3-ecf57b5a0470' in cluster 'default'
- ▼ 2016-02-11T20:31:02Z [INFO] Saving state! module="statemanager"

Container instance draining

There might be times when you need to remove a container instance from your cluster; for example, to perform system updates, update the Docker daemon, or to scale down the cluster capacity. Amazon ECS provides the ability to transition a container instance to a DRAINING status. This is referred to as *container instance draining*. When a container instance is set to DRAINING, Amazon ECS prevents new tasks from being scheduled for placement on the container instance. Any tasks that are part of a service that are in a PENDING state are stopped immediately. If there is available container instance capacity in the cluster, the service scheduler will start replacement tasks. If there isn't enough container instance capacity, a service event message will be sent indicating the issue.

Tasks that are part of a service on the container instance that are in a RUNNING state are transitioned to a STOPPED state. The service scheduler attempts to replace the tasks according to the service's deployment configuration parameters, `minimumHealthyPercent` and `maximumPercent`. For more information, see [Service definition parameters \(p. 534\)](#).

- If `minimumHealthyPercent` is below 100%, the scheduler can ignore `desiredCount` temporarily during task replacement. For example, `desiredCount` is four tasks, a minimum of 50% allows the scheduler to stop two existing tasks before starting two new tasks. If the minimum is 100%, the service scheduler can't remove existing tasks until the replacement tasks are considered healthy. If tasks for services that do not use a load balancer are in the RUNNING state, they are considered healthy. Tasks for services that use a load balancer are considered healthy if they are in the RUNNING state and the container instance they are hosted on is reported as healthy by the load balancer.
- The `maximumPercent` parameter represents an upper limit on the number of running tasks during task replacement, which enables you to define the replacement batch size. For example, if

`desiredCount` of four tasks, a maximum of 200% starts four new tasks before stopping the four tasks to be drained (provided that the cluster resources required to do this are available). If the maximum is 100%, then replacement tasks can't start until the draining tasks have stopped.

Any standalone tasks in the PENDING or RUNNING state are unaffected; you must wait for them to stop on their own or stop them manually.

A container instance has completed draining when all tasks running on the instance transition to a STOPPED state. The container instance remains in a DRAINING state until it is activated again or deleted. You can verify the state of the tasks on the container instance by using the [ListTasks](#) operation with the `containerInstance` parameter to get a list of tasks on the instance followed by a [DescribeTasks](#) operation with the Amazon Resource Name (ARN) or ID of each task to verify the task state.

When you are ready for the container instance to start hosting tasks again, you change the state of the container instance from DRAINING to ACTIVE. The Amazon ECS service scheduler will then consider the container instance for task placement again.

Draining container instances

You can use the [UpdateContainerInstancesState](#) API action or the `update-container-instances-state` command to change the status of a container instance to DRAINING.

The following steps can be used to set a container instance to draining using the AWS Management Console.

To set your container instance to DRAINING (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters** and select the cluster.
3. Choose the **ECS Instances** tab and select the check box for each container instance you want to drain.
4. Choose **Actions, Drain instances**.
5. After the instances are processed, choose **Done**.
6. When you want to activate the container instances again, repeat these same steps but choose **Activate instances** from the **Actions** menu.

Deregister an Amazon EC2 backed container instance

Important

This topic is for container instances created in Amazon EC2 only. For more information about deregistering external instances, see [Deregistering an external instance \(p. 421\)](#).

When you are finished with an Amazon EC2 backed container instance, you should deregister it from your cluster. Following deregistration, the container instance is no longer able to accept new tasks.

If you have tasks running on the container instance when you deregister it, these tasks remain running until you terminate the instance or the tasks stop through some other means. However, these tasks are orphaned which means they are no longer monitored or accounted for by Amazon ECS. If an orphaned task on your container instance is part of an Amazon ECS service, then the service scheduler starts another copy of that task, on a different container instance, if possible. Any containers in orphaned service tasks that are registered with a Classic Load Balancer or an Application Load Balancer target group are deregistered. They begin connection draining according to the settings on the load balancer

or target group. If an orphaned tasks is using the `awsvpc` network mode, their elastic network interfaces are deleted.

If you intend to use the container instance for some other purpose after deregistration, you should stop all of the tasks running on the container instance before deregistration. This stops any orphaned tasks from consuming resources.

When deregistering a container instance, be aware of the following considerations.

- Because each container instance has unique state information, they should not be deregistered from one cluster and re-registered into another. To relocate container instance resources, we recommend that you terminate container instances from one cluster and launch new container instances in the new cluster. For more information, see [Terminate your instance](#) in the *Amazon EC2 User Guide for Linux Instances* and [Launching an Amazon ECS container instance \(p. 364\)](#).
- If the container instance is managed by an Auto Scaling group or a AWS CloudFormation stack, terminate the instance by updating the Auto Scaling group or AWS CloudFormation stack. Otherwise, the Auto Scaling group or AWS CloudFormation will create a new instance after you terminate it.
- If you terminate a running container instance with a connected Amazon ECS container agent, the agent automatically deregisters the instance from your cluster. Stopped container instances or instances with disconnected agents are not automatically deregistered when terminated.
- Deregistering a container instance removes the instance from a cluster, but it does not terminate the Amazon EC2 instance. If you are finished using the instance, be sure to terminate it to stop billing. For more information, see [Terminate your instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

To deregister a container instance (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, choose the Region in which your container instance is registered.
3. In the navigation pane, choose **Clusters** and select the cluster that hosts the container instance.
4. On the **Cluster : name** page, choose the **ECS Instances** tab.

The screenshot shows the AWS Management Console interface for the ECS Instances tab. At the top, there are three tabs: Services, Tasks, and ECS Instances, with ECS Instances being the active tab. Below the tabs, there is a button to "Add additional ECS Instances using Auto Scaling or Amazon EC2". A "Filter in this page" input field is present. The main area displays a table titled "Viewing 1-1 Container Instance". The table has columns: Container Instance, EC2 Instance, Agent ..., Status, Available..., and Available... (partially visible). One row is shown, containing the ID "3de21d77-d1d7-4795-a3b3-ed6e6e5d7d353", the EC2 ID "i-501f2599", the status "true", the status "ACTIVE", and two partially visible values "2048" and "3955".

Container Instance	EC2 Instance	Agent ...	Status	Available...	Available...
3de21d77-d1d7-4795-a3b3-ed6e6e5d7d353	i-501f2599	true	ACTIVE	2048	3955

5. Select the container instance ID to deregister. This takes you to the container instance detail page.
6. On the **Container Instance : id** page, choose **Deregister**.
7. Review the deregistration message and choose **Deregister**.
8. If you are finished with the container instance, terminate the underlying Amazon EC2 instance. For more information, see [Terminate Your Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

Amazon ECS container agent

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

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

Note

For tasks using the Fargate launch type and platform version 1.3.0 and prior, the Amazon ECS container agent is installed on the AWS managed infrastructure used for these tasks. If you are only using tasks with the Fargate launch type, no additional configuration is needed and the content in this topic does not apply. For tasks using the Fargate and platform version 1.4.0 and later (for Linux) or 1.0.0 or later (for Windows), the Fargate container agent is used. For more information, see [AWS Fargate platform versions](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Topics

- [Installing the Amazon ECS container agent \(p. 431\)](#)
- [Amazon ECS container agent versions \(p. 440\)](#)
- [Updating the Amazon ECS container agent \(p. 448\)](#)
- [Amazon ECS container agent configuration \(p. 454\)](#)
- [Private registry authentication for container instances \(p. 468\)](#)
- [Automated task and image cleanup \(p. 471\)](#)
- [Amazon ECS container metadata file \(p. 472\)](#)
- [Amazon ECS task metadata endpoint \(p. 476\)](#)
- [Amazon ECS container agent introspection \(p. 503\)](#)
- [HTTP proxy configuration \(p. 504\)](#)

Installing the Amazon ECS container agent

If your container instance was not launched using an Amazon ECS-optimized AMI, you can install the Amazon ECS container agent manually using one of the following procedures. The Amazon ECS container agent is included in the Amazon ECS-optimized AMIs and does not require installation.

- For Amazon Linux 2 instances, you can install the agent using the `amazon-linux-extras` command. For more information, see [Installing the Amazon ECS container agent on an Amazon Linux 2 EC2 instance \(p. 432\)](#).
- For Amazon Linux AMI instances, you can install the agent using the Amazon YUM repo. For more information, see [Installing the Amazon ECS container agent on an Amazon Linux AMI EC2 instance \(p. 432\)](#).
- For non-Amazon Linux instances, you can either download the agent from one of the regional S3 buckets or from Docker Hub. If you download from one of the regional S3 buckets, you can optionally verify the validity of the container agent file using the PGP signature. For more information, see [Installing the Amazon ECS container agent on a non-Amazon Linux EC2 instance \(p. 433\)](#).

Note

The `systemd` units for both ECS and Docker services have a directive to wait for `cloud-init` to finish before starting both services. The `cloud-init` process is not considered finished until your Amazon EC2 user data has finished running. Therefore, starting ECS or Docker via Amazon EC2 user data may cause a deadlock. To start the container agent using Amazon EC2 user data you can use `systemctl enable --now --no-block ecs.service`.

Installing the Amazon ECS container agent on an Amazon Linux 2 EC2 instance

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

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

1. Launch an Amazon Linux 2 EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
2. Connect to your instance.
3. Disable the `docker` Amazon Linux extra repository. The `ecs` Amazon Linux extra repository ships with its own version of Docker, so the `docker` extra must be disabled to avoid any potential future conflicts. This ensures that you are always using the Docker version that Amazon ECS intends for you to use with a particular version of the container agent.

```
[ec2-user ~]$ sudo amazon-linux-extras disable docker
```

4. Install and enable the `ecs` Amazon Linux extra repository.

```
[ec2-user ~]$ sudo amazon-linux-extras install -y ecs; sudo systemctl enable --now ecs
```

5. (Optional) You can verify that the agent is running and see some information about your new container instance with the agent introspection API. For more information, see the section called ["Container agent introspection" \(p. 503\)](#).

```
[ec2-user ~]$ curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Note

If you get no response, ensure that you associated the Amazon ECS container instance IAM role when launching the instance. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

Installing the Amazon ECS container agent on an Amazon Linux AMI EC2 instance

To install the Amazon ECS container agent on an Amazon Linux AMI EC2 instance using the Amazon YUM repo, use the following steps.

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

1. Launch an Amazon Linux AMI EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
2. Connect to your instance.

3. Install the `ecs-init` package. For more information about `ecs-init`, see the [source code on GitHub](#).

```
[ec2-user ~]$ sudo yum install -y ecs-init
```

4. Start the Docker daemon.

```
[ec2-user ~]$ sudo service docker start
```

Output:

```
Starting cgconfig service: [ OK ]  
Starting docker: [ OK ]
```

5. Start the `ecs-init` upstart job.

```
[ec2-user ~]$ sudo start ecs
```

Output:

```
ecs start/running, process 2804
```

6. (Optional) You can verify that the agent is running and see some information about your new container instance with the agent introspection API. For more information, see the section called “Container agent introspection” (p. 503).

```
[ec2-user ~]$ curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Installing the Amazon ECS container agent on a non-Amazon Linux EC2 instance

To install the Amazon ECS container agent on a non-Amazon Linux EC2 instance, you can either download the agent from one of the regional S3 buckets or from Docker Hub. If you download from one of the regional S3 buckets, you can optionally verify the validity of the container agent file using the PGP signature.

Important

Downloading the ECS agent from Docker Hub will be subject to Docker Hub rate limits. The rate limits can be avoided by downloading the ECS agent directly from Amazon S3 rather than Docker Hub. For more information, see [Docker Hub - Download rate limit](#).

Note

When using a non-Amazon Linux AMI, your Amazon EC2 instance requires `cgroupfs` support for the `cgroup` driver in order for the ECS agent to support task level resource limits. For more information, see [ECS agent on GitHub](#).

The latest Amazon ECS container agent files, by Region, are listed below for reference.

Region	Region Name	Container agent	Container agent signature
us-east-2	US East (Ohio)	ECS container agent	PGP signature
us-east-1	US East (N. Virginia)	ECS container agent	PGP signature

Region	Region Name	Container agent	Container agent signature
us-west-1	US West (N. California)	ECS container agent	PGP signature
us-west-2	US West (Oregon)	ECS container agent	PGP signature
ap-east-1	Asia Pacific (Hong Kong)	ECS container agent	PGP signature
ap-northeast-1	Asia Pacific (Tokyo)	ECS container agent	PGP signature
ap-northeast-2	Asia Pacific (Seoul)	ECS container agent	PGP signature
ap-south-1	Asia Pacific (Mumbai)	ECS container agent	PGP signature
ap-southeast-1	Asia Pacific (Singapore)	ECS container agent	PGP signature
ap-southeast-2	Asia Pacific (Sydney)	ECS container agent	PGP signature
ca-central-1	Canada (Central)	ECS container agent	PGP signature
eu-central-1	Europe (Frankfurt)	ECS container agent	PGP signature
eu-west-1	Europe (Ireland)	ECS container agent	PGP signature
eu-west-2	Europe (London)	ECS container agent	PGP signature
eu-west-3	Europe (Paris)	ECS container agent	PGP signature
sa-east-1	South America (São Paulo)	ECS container agent	PGP signature
us-gov-east-1	AWS GovCloud (US-East)	ECS container agent	PGP signature
us-gov-west-1	AWS GovCloud (US-West)	ECS container agent	PGP signature

To install the Amazon ECS container agent on a non-Amazon Linux EC2 instance

1. Launch an Amazon EC2 instance with an IAM role that allows access to Amazon ECS. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
2. Connect to your instance.
3. Install the latest version of Docker on your instance.

Note

The Amazon Linux AMI always includes the recommended version of Docker for use with Amazon ECS. You can install Docker on Amazon Linux with the `sudo yum install docker -y` command.

4. Check your Docker version to verify that your system meets the minimum version requirement.

```
ubuntu:~$ sudo docker version
```

Output:

```
Client version: 1.4.1
Client API version: 1.16
```

```
Go version (client): go1.3.3
Git commit (client): 5bc2ff8
OS/Arch (client): linux/amd64
Server version: 1.4.1
Server API version: 1.16
Go version (server): go1.3.3
Git commit (server): 5bc2ff8
```

In this example, the Docker version is 1.4.1, which is below the minimum version of 1.9.0. This instance needs to upgrade its Docker version before proceeding. For information about installing the latest Docker version on your particular Linux distribution, go to <https://docs.docker.com/engine/installation/>.

5. Run the following commands on your container instance to allow the port proxy to route traffic using loopback addresses.

```
ubuntu:~$ sudo sh -c "echo 'net.ipv4.conf.all.route_localnet = 1' >> /etc/sysctl.conf"
ubuntu:~$ sudo sysctl -p /etc/sysctl.conf
```

6. Run the following commands on your container instance to enable IAM roles for tasks. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

```
ubuntu:~$ sudo apt-get install iptables-persistent
ubuntu:~$ sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT
--to-destination 127.0.0.1:51679
ubuntu:~$ sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j
REDIRECT --to-ports 51679
```

7. Add an iptables route to block off-host access to the introspection API endpoint.

```
ubuntu:~$ sudo iptables -A INPUT -i eth0 -p tcp --dport 51678 -j DROP
```

8. Write the new **iptables** configuration to your operating system-specific location.

- For Debian/Ubuntu:

```
sudo sh -c 'iptables-save > /etc/iptables/rules.v4'
```

- For CentOS/RHEL:

```
sudo sh -c 'iptables-save > /etc/sysconfig/iptables'
```

9. Create the /etc/ecs directory and create the Amazon ECS container agent configuration file.

```
ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
```

10. Edit the /etc/ecs/ecs.config file and add the following contents. If you do not want your container instance to register with the default cluster, specify your cluster name as the value for ECS_CLUSTER.

```
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
```

For more information about these and other agent runtime options, see [Amazon ECS container agent configuration \(p. 454\)](#).

Note

You can optionally store your agent environment variables in Amazon S3 (which can be downloaded to your container instances at launch time using Amazon EC2 user data). This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see [Storing Container Instance Configuration in Amazon S3 \(p. 467\)](#) and [Private registry authentication for tasks \(p. 300\)](#).

11. Pull and run the latest Amazon ECS container agent on your container instance.

Note

Use Docker restart policies or a process manager (such as **upstart** or **systemd**) to treat the container agent as a service or a daemon and ensure that it is restarted after exiting. For more information, see [Automatically start containers](#) and [Restart policies](#) in the Docker documentation. The Linux variants of the Amazon ECS-optimized AMI use the **ecs-init** RPM for this purpose, and you can view the [source code for this RPM](#) on GitHub.

The following example of the agent run command is broken into separate lines to show each option. For more information about these and other agent runtime options, see [Amazon ECS container agent configuration \(p. 454\)](#).

Important

Operating systems with SELinux enabled require the **--privileged** option in your **docker run** command. In addition, for SELinux-enabled container instances, we recommend that you add the **:Z** option to the **/log** and **/data** volume mounts. However, the host mounts for these volumes must exist before you run the command or you receive a **no such file or directory** error. Take the following action if you experience difficulty running the Amazon ECS agent on an SELinux-enabled container instance:

- Create the host volume mount points on your container instance.

```
ubuntu:~$ sudo mkdir -p /var/log/ecs /var/lib/ecs/data
```

- Add the **--privileged** option to the **docker run** command below.
- Append the **:Z** option to the **/log** and **/data** container volume mounts (for example, **--volume=/var/log/ecs/:/log:Z**) to the **docker run** command below.

- a. (Optional) Download the ECS container agent tarball from the regional S3 URL and load it. If you don't download the agent tarball from S3, the **docker run** command in the next step will download it from Docker Hub for you automatically.

```
ubuntu:~$ curl -o ecs-agent.tar https://s3.amazonaws.com/amazon-ecs-agent-us-east-1/ecs-agent-latest.tar
```

Note

To download other versions of the Amazon ECS container agent, use one of the following formats, changing the version number in the URL:

```
ecs-agent-<version>.tar  
ecs-agent-<SHA>.tar
```

For example:

```
https://s3.amazonaws.com/amazon-ecs-agent-us-east-1/ecs-agent-v1.18.0.tar
```

```
https://s3.amazonaws.com/amazon-ecs-agent-us-east-1/ecs-agent-c0defea9.tar
```

Load the ECS container agent image.

```
ubuntu:~$ sudo docker load --input ./ecs-agent.tar
```

- b. Run the ECS container agent image.

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run:/var/run \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
--volume=/etc/ecs:/etc/ecs \
--net=host \
--env-file=/etc/ecs/ecs.config \
amazon/amazon-ecs-agent:latest
```

Important

The host network mode is the only supported network mode for the container agent container. For more information, see [Running the Amazon ECS Container Agent with Host Network Mode \(p. 440\)](#).

Note

If you receive an `Error response from daemon: Cannot start container` message, you can delete the failed container with the `sudo docker rm ecs-agent` command and try running the agent again.

12. (Optional) If you downloaded the Amazon ECS container agent file from S3, you can verify the validity of the file.

- Download and install GnuPG. For more information about GNUpG, see the [GnuPG website](#). For Linux systems, install gpg using the package manager on your flavor of Linux.
- Retrieve the Amazon ECS PGP public key. You can use a command to do this or manually create the key and then import it.
 - Option 1: Retrieve the key with the following command.

```
gpg --keyserver htps://keys.openpgp.org --recv BCE9D9A42D51784F
```

- Option 2: Create a file with the following contents of the Amazon ECS PGP public key and then import it:

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

mQINBFq1SasBEAD1iGcT1NVJ1ydfN8DqebYYe9ne3dt6jqKFmKowLmm6LLGJe7HU
jGtqhCWRDkN+qPpHqdArRgDZAtn2pXY5fEipHgar4CP8QgRnRMO2f1741mavr4Vg
7K/KH8VHlq2uRw32/B94XLEgRbGTMdWFdKuxoPCttBQaMj3LGn6Pe+6xVWRkChQu
BoQAhjBQ+bEm0kNy0LjNgjNlnl3UMAG56t8E3LANIgGgEnpNsB1UwfWluPoGZoTx
N+6pHBJkKIL/1v/ETU4FXpYw2zvhWNahxeNRnoYj3uycHkeliCrw4kj0+skizBgO
2K7oVX8Oc3j5+ZilhL/qDLxmUCb2az5cMM1mOoF8EKX5HaNuq1KfwJxqXE6NNICo
1FTrT7QwD5fMNld3FanLgv/ZnIrsSaqJOL6zRSq8O4LN1OWBVbndExk2Kr+5kFx
51BPgfPgRj5hQ+KTHMa9Y8Z7yUc64BJiN6F9N17FJuSsfqbdkvRLsQRbcBG9qxX3
rJAEhieJzVMEUNl+EgeCkxj5xuSkNU7zw2c3hQZqEcrADLV+hvFJktOz9Gm6xzbg
1TnWWCz4xrIWtuEBA2qE+M1DheVd78a3gIsEaSTfQq0osYXaQbv1nSWOoc1y/5zb
zizHTJihLtUyls9WisP2s0emeHZicVMfW61EgPrJAIupgc7kyZvFt4YwfARAOAB
tCRBbWF6b24gRUNTDx1Y3Mtc2VjdXJpdH1AYW1hem9uLmNvbT6JAhwEEAECAAyF
AlrjL0YACgkQHivRXs0TaQrg1g/+JppwPqHn1VPmv7lessB8I5UqZeD6p6uVpHd7
```

```
Bs3pcPp8BV7BdRbs3sPLt5bV1+rKq0lw+0gZ4Q/ue/YbwTOAt4qY00cEo0HgcnaX
lsB827QfZIVtGWMhuh9xzm/SJkvngml6KB3YJNwF61A9qj37/VbVVLzvcmaZ
McWB4HUMNrhd0JgBCoogIpqCbpJEvUc02Bjn23eEJsS9kC7OUAHyQkVnx4d9UzXF
4OoISF6hmQKIBoLnRrAlj5Qvs3GhvHQ0ThYq0Grk/KMJXJX2CSqt7tWJ8gk1n3H3Y
SReRXJRnv7DsDDBwFgt6r5Q2HW1TBuva0zy5hF6maD09nHcNnvBjqADzeT8Tr/Qu
bBCLzkNSYqqkpgtw7seoD2P4n1giRvDAOEfmZpVkr+C252IaH1HZFEz+TvbVQM
Y8OWWxmIJW+j6evjo3N1e019Uh71jvoF8z1ljb14bsL2c+QTJmOv7nRqzDQgCWyp
Id/v2dUVVTk1j9omuLBbwNjzQCB+72lcIzJhYmaPIHC4LcKQG+/f4lexuItena+k
1EJQhYtyvxcb1h6Yn/wzNg2NWOb3vqY/F7m69ixAwgt1MgPCDE4aJ86zrrXYFz
N2HqkTSQh77Z8KPkmhGopsnM/reMu1PdInB249nA0dzOn+nj+tFOYClalaFyjs
Z0r1QAOAjKEECACMFAlq1SasCGwMHcwkIBwMCAQYVCAIJCgsEFgIDAQIEAQIX
gAAKCRC86dmkLVF4T9iFEACEenkmlsWuX34R3c0vamHrPxvfkyI1f1EUen8D1h
ux9xy6jCEROHWeP0rjGK4QDPGm93sWJ+s1UAKg214QRVzft0y9/Ddr+twApA0fzy
uavIthGd+03jAAo6udYDE+cZC3P7XbbDiYEwk4XAF9I1jB8hTZUgvXBL046jhG
eM17+crgUyQeetki0QemLbsbXQ40Bd9V7zf7XJraFd8VrvNUwNb+9KftgAsc9rk+
YIT/PEf+YOPysgcxi4sTbWgthyCulVnuGoskgDv4v73PALU0ieUrVvvQVqWMRvhVx1
0X90J7c1C1OyhleEQQ1aFTgmQjmXexVTwIBm8LvysFK6YXM41KjOr1z3+6xBIm/qe
bFyLUnf4WoiuOplaAhk9pRY+XEnGNxdtN4D26Kd0F+PLkm3Tr3Hy3b1Ok34F1Gr
KVHUq1TzD7cvMnnNKEELTucKX+1mV3an16nmAg/my1JSUT6BNK2JpY1s/kkSGSE
XQ4zuF2IGCpvBFhYAlt5un5zwqkwQR3/n2kwAoDzonJcehdw/C/cGos5D0aIU7I
K2X2aTD3+pA7Mx3IMe2hqmYqrT9X4yF1P1EVreBRJ3HDezAgJrNh0GOWRQkhIx
gz6/cTR+ekr5TptVsZs9few2GpI5bCgBKBSzIissT89aw7mAKWut0Gcm4qM9/yK6
1bkCDQRatUmrARAAXNPvvwreJ2yAiFcUpdRlvhsuOgnxvs1QgsIw3H7+Pacr9Hpe
8uftYZqdC82KeSKhpHq7c8gMTMucIINtH25x9BCC73E33EjCL9Lqv1TL7+QkgHe
T+J1hZwdD8Mx2K+Lvvvu/aWkNrfMuNwyDuciSI4D5QHa8T+F8fgN40Tpwyjirzel
5yoICMr9hvcbzDNv/oZKcxjx+XKgnFc3wrnDfJfntfDAT7ecwbUTL+viOKJ646s+
psiqXRytVvYInEhLvrJ0aV6zHfoigE/Bils6/g7ru1Q6CEHqEw++APs5CcE8VzJu
WAGSVHZgun5Y9N4quR/M9Vm+IPMhTxrAg7rOvyRN9cAxfeSMf77I+XTifigNna8x
t/ModjXr1fjF4pThEi5u6WsujRdfwJy2azEv3vevodTi4HoJReH6dFRa6y8c+UDg1
2iHiOKIp0qLbHEf0mHcd2fix+AaJKMnPgnku9qCFEMbgSRJpXz6bfwnY1QuKE+i
R6ja0frUnt2jhiGG/F8RceXzohaaC/Cx7LUUCUFWc0n7z32C9/Dtj7I1PMoacdZzz
bjjZrKO/ZDv+UN/c9dwAk1lzaYPMwGbkuay68EBstn1liW34aWm61iHhxioVPKSp
VJfyiXPO0EXqujthLAeChfjcnS1I2YshT1dv2PafG53fp33ZdzeUgsBo+EAEQEA
AYkCHwQYAQIAQCUCWrVJqwlBAAKCRc86dmkLVF4T+ZdD/9x/8APzgNjf3o3StrF
jvnV1ycyhwYGAeBjiu7wjsNNWzMFov15tLjB7AqeVxz+nWKDD/mIQ45OZvnYZuy
X7DR0Jszah9wrYTxZLvrAu+t6UL0y/XQ4L1GZ9QR6+r+7t1Mvbfy7B1HbvX/qYt
Rwe/uwdibIOcagEzyX+2D3kT01HO5XThbXaNf8AN8zha91Jt2Q2UR2X5T6JcwtMz
FBvZn1LSmZyE0EQehS2iUur4uWOpGppuqvnb0jbcvCHKgDGrqZ0smKNAQng54
F365W3g8AfY48s8XQwzmccliowyX9bt8PZiEi0J4QmQh0aXkpqZyFefuWeOL2R94S
XKzr+gRh3BAUloqF+qK+IUMxTip9KTPNvYDpiC66yBiT6gFDji5Ca9pGpJXrC3xe
TXikQ8DBWDhBPVPPrulIaenTtzeOsPc4I85yt5U9roPStcOr343w5yEaJagt6S
Gc5r9ysjkfH6+6rb1lujxMgROSqtgr+RyB+v9A5/OgtNzc811K6u4uoOCde8jUUW
vqWKvjjB/Kz3u4zaeNu2ZyyHaQoUh+TETCw+jsy91hbEzqN5yQYGi4pVmDkY5vu
1XbJnbqPKpRxf9BecV9AmBPgdbDq/5LnHJXg+G8YQOgp4lR/hC1TEFdip5wM8AK
CWsENyt2o1rjgMXiZOMF8A5oBLkCDQRatUoSARAAr77kj7j2QR2SzEOS1FBvV7oS
mFeSNz9xZssqrsm6btwsHm6YLDwc7Sdf2esDdyzONETwqrVCg+FxgL8hmo9hS4c
rR6tmrP0m0mptr+xLLsKcaP7ogIXsyZnrEAESvW8PnfayoiPCdc3cMCR/1TnHFGA
7EuR/XLBmi7Qg9tByVYQ5Yj5wB9V4B2yeCt3XtzPqeLkvaxl7PNelaHGJQY/xo+m
V0bndxf9IY+4oFJ4b1D32WqvyxEso7vW6WBh7oqv3Zbm0yQrr8a6mDBpqLkvWwNI
3kpJR974tg5o5LfDu1BeeyHWPSSGM4u/G4JB+JIG1Ady+RmoWEt4BqTCZ/knnoGvw
D5sTCxbKdmu0mhGyTssog+300cGYHV7pWYPhazKHMPm201xKCjH1RfzRULzGKjD+
yMLT1I3AXFmLmZJXk1o1vE3/wgMqCXscbycbLjLD/bxiuFWo3rzoezeXjgi/DJx
jKBAyBTY05nMcth109oaFd9d0HbsoudkIMnsgGBE766Piro6MHoOT0rX107Tp4pI
rwuSOsc6XzCzdImj0Wc6axS/HeUKRXdJXwno5awTwXKRJMxGfhCvSvbcbc2Wx+L
IKvmB7EB4K3fmjFFE67yolmiw2qRcUbFygtH3eL5XZU28MiCpue8Y8GKj0BAUyvf
KeM1rO8Jm3iRac5a/D0AEQEAAYkEPgQYAQIAQCUCWrVLkgIbAgIpCRC86dmkLVF4
T8FdIAQZAQIABgUCWrVLkgAKCRDePL1hra+LjtHYD/9MucxdFe6bX01dQR4tKhhQ
P0LRqy6z1BY9ILCLOWndGZdqorogUiUymgn3VhEHvtxToohcN7qouM01PnsRnOeS
EYjf8Xrb1clzkd6xULwm0c1Tb9bBxnBc/4PFvHABzW3QzusaZniNgkuxt6BtfloS
Of4inq71kjmgK+T1zQ6mUMQUG228NUQC+a84EPqYyAeY1sgvgB7hJBhYL0QAxhcW
6m20Rd8iEc6HyzJ3yCOCsKip/nRWAf00vFhFRBp0+m0ZwnJM8cPRFj0qqzFpKH9
HpDmTrC4wKp1+TL52LyEqNh4yZitXmZNV7giSRIkk0eDSko+bFy6VbMzKUMkUJK3
D3eHFAMkujmbfJmSMTJ0PGn5SB1HyjCZNx6bhIIbQyEUB9gKCMuFaQXKwKpF6rj0
iQXAJxLR/shZ5Rk96VxzOphU17T90m/PnUEEPwq8KsBhnMRgxa0RFidDP+n9fgtv
HLmr0qX9zBCVXh0mdWYLRWvmzQFWzG7AoE55fkf8nAEPsalrCdtaNUBHRXA0OQxG
AHMODJQqvbSmqMvuAdjkDWpFu5y0My5ddU+hiUzUyQLjL5Hhd5LOUDdewlZgIw1j
```

Amazon Elastic Container Service Developer Guide
Installing the Amazon ECS container agent
on a non-Amazon Linux EC2 instance

```
xrEAUzDKetnemM8GkHxDgg8koev5frmShJuCe7vSjKpCNg3EIJSqqMOPFjJuLwtZ
vjHeDNbJy6uNL65ckJy6WhGjEADS2WAW1D6Tfekkc21ssIXk/LqEpLMR/0g5OUif
wcEN1rS9IJXBwIy8Me1N9qr5KcKQlMfdfBNEyyceBhyv10MDyHOKC+7PofMtKGBq
13QieRHv5GJ8LB3fc1qHV8pwTT03Bc8z2g0TjmUYAN/ixETdReDoKavWJYSE9yoM
aaJu279iovTrwpECse0XkiRyKTtjwOb73CGkBZpJyqux/rmCV/fp4ALdSW8zbz
FJVORaivh0WwzjpfQKhwcU91ABXi2UvVm14v0AfE17oiJPSU1zM4fEny4oiIBX1R
zhFNih1UjIu82X16mTm3BwbIga/s1fnQRGzyhqUIMi+mWra23EwjChaxpvjjcUH
5illLc5Zq781aCYRygYQw+hu5nfkOH1R+Z50Ubxd/aQufnGIAx7kPMD3Lof4K1dD
Q8ppQriUvxVo+4nPv6rpTy/PyqCLWDjkguHpJsEFsMkwajrAz0QNSAU5CJ0G2Zu4
yxvYlumHCE17nbFrm0v1ia75sa8KnywTDsyZsU3CxOcf3g+g1xWTpjJqy2bYXlqz
9uDOWtArWHOis6bq819RE6xr1RBVXS6uqgQIZFBGyq66b0dIq4D2JdsUvgEMaHbc
e7tBfeB1CMBdA64e9Rq7bFR7Tvt8gasCZY1Nr3lydh+dFHIEkH53HzQe6188HEic
+0jVnLkCDQRa55wJARAyLya2Lx6gyoWoJN1a6740q3o8e9d4KggQfGMTCf1meq
ivuzgN+3DZHN+9ty2KxXMtn0mhHBerZdbNjyjmNT1gAgrhPNB4HXBxUm2wS57WK
DNmade914L7FWTPAWBG2Wn4480EHTqsC1ICXXW91IICgc1AEyIq0Yq5mAdTEgRJS
Z8t4GpwtDL9gNQyFXaWQmDmkAsCygQMvhAlmu9xO1zQG5CxSnZFK7zcuL60k14Z3
Cmt49k4T/7ZU8goWi8tt+rU78/IL3J/fF9+1civ1OwuUiDgfPCSVouW1JojsdCQA
L+RZJcoXq71fOfJ/eNjeOSstCTDPfTCL+kThE6E5neDtboQHBYkEX1BRiTedsV4+M
ucgiTrdQFWkf89G72xdv8ut9AYYQ2BbEYU+JAYhUh8rYYui2dHKU1gjNvJscuUwb
+QEeqJIRleJRhrO+/CHgMs4fZAkWF1VfhKBkcKmEjLn1f7EJJUUW842hKXj0/AUPX
1CHsNjziRceuJCJYox1cwsoq61te50GzqCIxTn9xUc0UMKFeggNAFys1K+TDTm3
Bzo8H5ucjCUEmUm91hkGwqTZg0lRX5eqPx+JBoSaObqhgqCa5IPinKRa6MgoFPHK
6sYKqroYwBGgZm6Jss5chpNchvJMs/3WXNOEVg0J3z3vP0DMhxqWm+r+n9z1w8qsa
EQEEAAyKEPgQYAQgACQUCWuecCQ1bAgIpCRC86dmkLvf4T8FdIAQZAQgABgUCWuec
CQAKCRBQ3szEcQ5hr+ykD/4t0LRHFHXuKUCxgGaubUcvtsFrwBKmalcyjqapms8u
6Sk0wfGRI32G/GhOrp0Ts/MOkbObq6VLTh8N5Yc/53ME18zQFw9Y5AmRoW4PZXER
ujss5s7p4oR7xHMihMjCCBn1bvrR+34YPfgzTcgLiOEHYT8UTxwnGmXOvNkMM7md
xD3CV5q6VAtE8WKBo/2201I3fcQlc9r/oWX4kXXkb0v9hoGwKbDJ1tzqtPrp/xFt
yohqnvImpnlz+Q9zXmbnwYL9/g8VCmW/NN2gju2G3Lu/T1FUWIT4v/5OPK6TdeNb
VKJO4+S8bTayqSG9CMl1S57KsgCo5HuHQWeSNHI+fpe5ox6FALPT9JLDce8OZzli
cZZ0MELP37mOOQun0AlmHm/hVzf0f311PtbczqWaE51tJvgUR/nZFo6Ta305Ezs
3V1EJNQ11jf/6DH87SxvAoRIARCuZd0qxBCDK0avpFzUtbJd241RA3WJpkEiMqKv
RDVZkE4b6TWf0o+LaVfK6E8oLpixegS4figC16mFrOdyRk+RJJf1Uyz0WTDVmt
g0U1CO1ezokMSqkJ7724pyjr2xf/r9/sC6aOJwB/lKgZkJfC6Nql7TlxVA31dUga
LEOvEJTT84gl+tYtfscDvALCtqL0jduSkUo+RXcB1tmXhA+tShWOpbS2Rtx/ixua
KohVD/0R4QxiSwQmICNtm9mw9ydi1lyjYXX5a9x4wMJracNY/LbybJPFnZnT4dYR
z4XjqysDwvvYZByaWoIe3QxjX84V6M1I2IdAT/xImu8gbaCI8tmyfpIrLnPKiR9D
VFYfGBxuAX7+HgPPSFtrHQONCALxxz1bNPs+zxt9r0M1lgcLyspWxSdmoYGZ6nQP
RO5Nm/ZVs+u2imPCRzNUZEMa+d1E6kHx0rS0dPiuj407NtPeYDkk0QtNagspsDvh
cK7CSqAiKmQ06UBTxqlTSRkm62e0Octcs3p30eHu5GRZf1uzTET0ZxYkaPgdrQknx
ozjP5mC7x+451cFcmcV94TFNL5hwEUUVJpmOgmzILCI8yoDTWzloo+i+fPFsXX4f
kynhE83mSEcr5VHFYrTY3mQXGmNj3bCluc/jq7ysGq69xiKmTlUeXFm+aojcR05i
zyShIRJZ0GZfuzDYFDmMV9amA/YQGygLw/+zP5ju5SW26dNxlf3MdFQE5JJ86rn9
MgZ4gcpazHEVUsbZsgkLizRp9imUiH8ymLqAXnfRGlU/LpNSefnvDFTtEIRcpOHC
bhayG0bk51Bd4mioOXnisK4j63nJXA27x5EVVHQ1sYRN8Ny4Fdr2tMAmj20+x+j
qX2yy/UX5nSPU492e2CdZ1UhoU0SRFY3bxKHKB7SDbVeav+K5g==
=Gi5D
-----END PGP PUBLIC KEY BLOCK-----
```

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

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

Import the Amazon ECS PGP public key with the following command.

```
gpg --import <public_key_filename>
```

- c. Download the ECS container agent signature. ECS container agent signatures are ASCII detached PGP signatures stored in files with the extension .asc. The signatures file has the same name as its corresponding executable, with .asc appended.

Substitute REGION with the Region that hosts the S3 bucket, for example us-east-1.

```
curl -o ecs-agent.asc https://s3.amazonaws.com/amazon-ecs-agent-REGION/ecs-agent-latest.tar.asc
```

- d. Verify the signature.

```
gpg --verify ecs-agent.asc ./ecs-agent.tar
```

Expected output:

```
gpg: Signature made Wed 16 May 2018 08:21:06 PM UTC using RSA key ID 710E61AF
gpg: Good signature from "Amazon ECS <ecs-security@amazon.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:           There is no indication that the signature belongs to the owner.
Primary key fingerprint: F34C 3DDA E729 26B0 79BE  AEC6 BCE9 D9A4 2D51 784F
Subkey fingerprint: D64B B6F9 0CF3 77E9 B5FB  346F 50DE CCC4 710E 61AF
```

Note

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

Running the Amazon ECS Container Agent with Host Network Mode

When running the Amazon ECS container agent, `ecs-init` will create the container agent container with the host network mode. This is the only supported network mode for the container agent container.

This enables you to block access to the [Amazon EC2 instance metadata service endpoint](#) (`http://169.254.169.254`) for the containers started by the container agent. This ensures that containers cannot access IAM role credentials from the container instance profile and enforces that tasks use only the IAM task role credentials. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

This also makes it so the container agent doesn't contend for connections and network traffic on the `docker0` bridge.

Amazon ECS container agent versions

Each Amazon ECS container agent version supports a different feature set and provides bug fixes from previous versions. When possible, we always recommend using the latest version of the Amazon ECS container agent. To update your container agent to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

Launching your container instances from the most recent Amazon ECS-optimized Amazon Linux 2 AMI ensures that you receive the current container agent version. To launch a container instance with the latest Amazon ECS-optimized Amazon Linux 2 AMI, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

To install the latest version of the Amazon ECS container agent on another operating system, see [Installing the Amazon ECS container agent \(p. 431\)](#). The table in [Linux Amazon ECS-optimized AMIs versions \(p. 342\)](#) shows the Docker version that is tested on Amazon Linux 2 for each agent version. The table in [Amazon ECS-Optimized Amazon Linux AMI Container Agent Versions \(p. 444\)](#) shows the Docker version that is tested on the Amazon Linux AMI for each agent version.

To see which features and enhancements are included with each agent release, see <https://github.com/aws/amazon-ecs-agent/releases>.

Amazon ECS-Optimized Amazon Linux 2 AMI Container Agent Versions

The Amazon ECS-optimized Amazon Linux 2 AMI comes prepackaged with the Amazon ECS container agent, Docker, and the `ecs-init` systemd service that controls the starting and stopping of the agent at boot and shutdown. The following table lists the container agent version, the `ecs-init` version, and the Docker version that is tested and packaged with each Amazon ECS-optimized Amazon Linux 2 AMI.

Note

As new Amazon ECS-optimized Amazon Linux 2 AMIs and Amazon ECS agent versions are released, older versions are still available for launch in Amazon EC2. However, we encourage you to [update to the latest version \(p. 448\)](#) of the Amazon ECS agent and to keep your container instance software up to date. If you request support for an older version of the Amazon ECS agent through AWS Support, you may be asked to move to the latest version as a part of the support process.

Important

Amazon ECS agent versions 1.20.0 and later have deprecated support for Docker versions older than 1.9.0.

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	<code>ecs-init</code> version
20211103	1.57.0	20.10.7	1.57.0-1
20211020	1.56.0	20.10.7	1.56.0-1
20211013	1.55.5	20.10.7	1.55.5-1
20210929	1.55.4	20.10.7	1.55.4-1
20210922	1.55.3	20.10.7	1.55.3-1
20210907	1.55.2	19.03.13-ce	1.55.2-1
20210819	1.55.1	19.03.13-ce	1.55.1-1
20210805	1.55.0	19.03.13-ce	1.55.0-1
20210802	1.54.1	19.03.13-ce	1.54.1-1
20210723	1.54.1	19.03.13-ce	1.54.1-1
20210708	1.54.0	19.03.13-ce	1.54.0-1
20210623	1.53.1	19.03.13-ce	1.53.1-1
20210609	1.53.0	19.03.13-ce	1.53.0-1
20210520	1.52.2	19.03.13-ce	1.52.2-1

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20210514	1.52.1	19.03.13-ce	1.52.1-1
20210504	1.52.0	19.03.13-ce	1.52.0-1
20210428	1.52.0	19.03.13-ce	1.52.0-1
20210413	1.51.0	19.03.13-ce	1.51.0-1
20210331	1.51.0	19.03.13-ce	1.51.0-1
20210316	1.50.3	19.03.13-ce	1.50.3-1
20210301	1.50.2	19.03.13-ce	1.50.2-1
20210219	1.50.2	19.03.13-ce	1.50.2-1
20210210	1.50.1	19.03.13-ce	1.50.1-1
20210202	1.50.0	19.03.13-ce	1.50.0-1
20210121	1.50.0	19.03.13-ce	1.50.0-1
20210106	1.49.0	19.03.13-ce	1.49.0-1
20201209	1.48.1	19.03.13-ce	1.48.1-1
20201130	1.48.1	19.03.13-ce	1.48.1-1
20201125	1.48.1	19.03.6-ce	1.48.1-1
20201119	1.48.0	19.03.6-ce	1.48.0-1
20201028	1.47.0	19.03.6-ce	1.47.0-1
20201013	1.46.0	19.03.6-ce	1.46.0-1
20200928	1.45.0	19.03.6-ce	1.45.0-1
20200915	1.44.4	19.03.6-ce	1.44.4-1
20200905	1.44.3	19.03.6-ce	1.44.3-1
20200902	1.44.3	19.03.6-ce	1.44.3-1
20200827	1.44.2	19.03.6-ce	1.44.2-1
20200820	1.44.1	19.03.6-ce	1.44.1-1
20200813	1.44.0	19.03.6-ce	1.44.0-1
20200805	1.43.0	19.03.6-ce	1.43.0-1
20200723	1.42.0	19.03.6-ce	1.42.0-1
20200708	1.41.1	19.03.6-ce	1.41.1-2
20200706	1.41.1	19.03.6-ce	1.41.1-1
20200623	1.41.0	19.03.6-ce	1.41.0-1

Amazon ECS-optimized Amazon Linux 2 AMI	Amazon ECS container agent version	Docker version	ecs-init version
20200603	1.40.0	19.03.6-ce	1.40.0-1
20200430	1.39.0	19.03.6-ce	1.39.0-1
20200402	1.39.0	18.09.9-ce	1.39.0-1
20200319	1.38.0	18.09.9-ce	1.38.0-1
20200218	1.37.0	18.09.9-ce	1.37.0-2
20200205	1.36.2	18.09.9-ce	1.36.2-1
20200115	1.36.1	18.09.9-ce	1.36.1-1
20200108	1.36.0	18.09.9-ce	1.36.0-1
20191212	1.35.0	18.09.9-ce	1.35.0-1
20191114	1.33.0	18.06.1-ce	1.33.0-1
20191031	1.32.1	18.06.1-ce	1.32.1-1
20191014	1.32.0	18.06.1-ce	1.32.0-1
20190925	1.32.0	18.06.1-ce	1.32.0-1
20190913	1.31.0	18.06.1-ce	1.31.0-1
20190815	1.30.0	18.06.1-ce	1.30.0-1
20190709	1.29.1	18.06.1-ce	1.29.1-1
20190614	1.29.0	18.06.1-ce	1.29.0-1
20190607	1.29.0	18.06.1-ce	1.29.0-1
20190603	1.28.1	18.06.1-ce	1.28.1-2
20190510	1.28.0	18.06.1-ce	1.28.0-1
20190402	1.27.0	18.06.1-ce	1.27.0-1
20190301	1.26.0	18.06.1-ce	1.26.0-1
20190215	1.25.3	18.06.1-ce	1.25.3-1
20190204	1.25.2	18.06.1-ce	1.25.2-1
20190127	1.25.1	18.06.1-ce	1.25.1-1
20190118	1.25.0	18.06.1-ce	1.25.0-1
20190107	1.24.0	18.06.1-ce	1.24.0-1
20181112	1.22.0	18.06.1-ce	1.22.0-1
20181016	1.20.3	18.06.1-ce	1.21.0-1

For more information about the Amazon ECS-optimized Amazon Linux 2 AMI, including AMI IDs for the latest version in each Region, see [Amazon ECS-optimized AMI \(p. 333\)](#).

Amazon ECS-Optimized Amazon Linux AMI Container Agent Versions

The Amazon ECS-optimized Amazon Linux AMI comes prepackaged with the Amazon ECS container agent, Docker, and the `ecs-init` service that controls the starting and stopping of the agent at boot and shutdown. The following table lists the container agent version, the `ecs-init` version, and the Docker version that is tested and packaged with each Amazon ECS-optimized AMI.

Note

As new Amazon ECS-optimized Amazon Linux AMIs and Amazon ECS agent versions are released, older versions are still available for launch in Amazon EC2. However, we encourage you to [update to the latest version \(p. 448\)](#) of the Amazon ECS agent and to keep your container instance software up-to-date. If you request support for an older version of the Amazon ECS agent through AWS Support, you may be asked to move to the latest version as a part of the support process.

Important

Amazon ECS agent versions 1.20.0 and later have deprecated support for Docker versions older than 1.9.0.

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	<code>ecs-init</code> version
2018.03.20210923	1.51.0	20.10.7	1.51.0-1
2018.03.20210723	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210519	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210413	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210331	1.51.0	19.03.13-ce	1.51.0-1
2018.03.20210316	1.50.3	19.03.13-ce	1.50.3-1
2018.03.20210301	1.50.2	19.03.13-ce	1.50.2-1
2018.03.20210219	1.50.2	19.03.13-ce	1.50.2-1
2018.03.20210210	1.50.1	19.03.13-ce	1.50.1-1
2018.03.20210202	1.50.0	19.03.13-ce	1.50.0-1
2018.03.20210121	1.50.0	19.03.13-ce	1.50.0-1
2018.03.20210106	1.49.0	19.03.13-ce	1.49.0-1
2018.03.20201209	1.48.1	19.03.13-ce	1.48.1-1
2018.03.20201130	1.48.1	19.03.13-ce	1.48.1-1
2018.03.20201125	1.48.1	19.03.6-ce	1.48.1-1
2018.03.20201119	1.48.0	19.03.6-ce	1.48.0-1
2018.03.20201028	1.47.0	19.03.6-ce	1.47.0-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2018.03.20201013	1.46.0	19.03.6-ce	1.46.0-1
2018.03.20200928	1.45.0	19.03.6-ce	1.45.0-1
2018.03.20200915	1.44.4	19.03.6-ce	1.44.4-1
2018.03.20200905	1.44.3	19.03.6-ce	1.44.3-1
2018.03.20200902	1.44.3	19.03.6-ce	1.44.3-1
2018.03.20200827	1.44.2	19.03.6-ce	1.44.2-1
2018.03.20200820	1.44.1	19.03.6-ce	1.44.1-1
2018.03.20200813	1.44.0	19.03.6-ce	1.44.0-1
2018.03.20200805	1.43.0	19.03.6-ce	1.43.0-1
2018.03.20200723	1.42.0	19.03.6-ce	1.42.0-1
2018.03.20200708	1.41.1	19.03.6-ce	1.41.1-2
2018.03.20200706	1.41.1	19.03.6-ce	1.41.1-1
2018.03.20200623	1.41.0	19.03.6-ce	1.41.0-1
2018.03.20200603	1.40.0	19.03.6-ce	1.40.0-1
2018.03.20200430	1.39.0	19.03.6-ce	1.39.0-1
2018.03.20200402	1.39.0	18.09.9-ce	1.39.0-1
2018.03.20200319	1.38.0	18.09.9-ce	1.38.0-1
2018.03.20200218	1.37.0	18.09.9-ce	1.37.0-2
2018.03.20200205	1.36.2	18.09.9-ce	1.36.2-1
2018.03.20200115	1.36.1	18.09.9-ce	1.36.1-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20200108	1.36.0	18.09.9-ce	1.36.0-1
2018.03.20191212	1.35.0	18.09.9-ce	1.35.0-1
2018.03.20191114	1.33.0	18.06.1-ce	1.33.0-1
2018.03.20191031	1.32.1	18.06.1-ce	1.32.1-1
2018.03.20191016	1.32.0	18.06.1-ce	1.32.0-1
2018.03.20191014	1.32.0	18.06.1-ce	1.32.0-1
2018.03.y	1.32.0	18.06.1-ce	1.32.0-1
2018.03.x	1.31.0	18.06.1-ce	1.31.0-1
2018.03.w	1.30.0	18.06.1-ce	1.30.0-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2018.03.v	1.29.1	18.06.1-ce	1.29.1-1
2018.03.u	1.29.0	18.06.1-ce	1.29.0-1
2018.03.t	1.29.0	18.06.1-ce	1.29.0-1
2018.03.s	1.28.1	18.06.1-ce	1.28.1-2
2018.03.q	1.28.0	18.06.1-ce	1.28.0-1
2018.03.p	1.27.0	18.06.1-ce	1.27.0-1
2018.03.o	1.26.0	18.06.1-ce	1.26.0-1
2018.03.n	1.25.3	18.06.1-ce	1.25.3-1
2018.03.m	1.25.2	18.06.1-ce	1.25.2-1
2018.03.l	1.25.1	18.06.1-ce	1.25.1-1
2018.03.k	1.25.0	18.06.1-ce	1.25.0-1
2018.03.j	1.24.0	18.06.1-ce	1.24.0-1
2018.03.i	1.22.0	18.06.1-ce	1.22.0-1
2018.03.h	1.21.0	18.06.1-ce	1.21.0-1
2018.03.g	1.20.3	18.06.1-ce	1.20.3-1
2018.03.f	1.20.2	18.06.1-ce	1.20.2-1
2018.03.e	1.20.1	18.03.1-ce	1.20.1-1
2018.03.d	1.20.0	18.03.1-ce	1.20.0-1
2018.03.c	1.19.1	18.03.1-ce	1.19.1-1
2018.03.b	1.19.0	18.03.1-ce	1.19.0-1
2018.03.a	1.18.0	17.12.1-ce	1.18.0-1
2017.09.l	1.17.3	17.12.1-ce	1.17.3-1
2017.09.k	1.17.2	17.12.0-ce	1.17.2-1
2017.09.j	1.17.2	17.12.0-ce	1.17.2-1
2017.09.i	1.17.1	17.09.1-ce	1.17.1-1
2017.09.h	1.17.0	17.09.1-ce	1.17.0-2
2017.09.g	1.16.2	17.09.1-ce	1.16.2-1
2017.09.f	1.16.1	17.06.2-ce	1.16.1-1
2017.09.e	1.16.1	17.06.2-ce	1.16.1-1
2017.09.d	1.16.0	17.06.2-ce	1.16.0-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	ecs-init version
2017.09.c	1.15.2	17.06.2-ce	1.15.1-1
2017.09.b	1.15.1	17.06.2-ce	1.15.1-1
2017.09.a	1.15.0	17.06.2-ce	1.15.0-4
2017.03.g	1.14.5	17.03.2-ce	1.14.5-1
2017.03.f	1.14.4	17.03.2-ce	1.14.4-1
2017.03.e	1.14.3	17.03.1-ce	1.14.3-1
2017.03.d	1.14.3	17.03.1-ce	1.14.3-1
2017.03.c	1.14.3	17.03.1-ce	1.14.3-1
2017.03.b	1.14.3	17.03.1-ce	1.14.3-1
2016.09.g	1.14.1	1.12.6	1.14.1-1
2016.09.f	1.14.0	1.12.6	1.14.0-2
2016.09.e	1.14.0	1.12.6	1.14.0-1
2016.09.d	1.13.1	1.12.6	1.13.1-2
2016.09.c	1.13.1	1.11.2	1.13.1-1
2016.09.b	1.13.1	1.11.2	1.13.1-1
2016.09.a	1.13.0	1.11.2	1.13.0-1
2016.03.j	1.13.0	1.11.2	1.13.0-1
2016.03.i	1.12.2	1.11.2	1.12.2-1
2016.03.h	1.12.1	1.11.2	1.12.1-1
2016.03.g	1.12.0	1.11.2	1.12.0-1
2016.03.f	1.11.1	1.11.2	1.11.1-1
2016.03.e	1.11.0	1.11.2	1.11.0-1
2016.03.d	1.10.0	1.11.1	1.10.0-1
2016.03.c	1.10.0	1.11.1	1.10.0-1
2016.03.b	1.9.0	1.9.1	1.9.0-1
2016.03.a	1.8.2	1.9.1	1.8.2-1
2015.09.g	1.8.1	1.9.1	1.8.1-1
2015.09.f	1.8.0	1.9.1	1.8.0-1
2015.09.e	1.7.1	1.9.1	1.7.1-1
2015.09.d	1.7.1	1.9.1	1.7.1-1

Amazon ECS-optimized Amazon Linux AMI	Amazon ECS container agent version	Docker version	<code>ecs-init</code> version
2015.09.c	1.7.0	1.7.1	1.7.0-1
2015.09.b	1.6.0	1.7.1	1.6.0-1
2015.09.a	1.5.0	1.7.1	1.5.0-1
2015.03.g	1.4.0	1.7.1	1.4.0-2
2015.03.f	1.4.0	1.6.2	1.4.0-1
2015.03.e	1.3.1	1.6.2	1.3.1-1
2015.03.d	1.2.1	1.6.2	1.2.0-2
2015.03.c	1.2.0	1.6.2	1.2.0-1
2015.03.b	1.1.0	1.6.0	1.0-3
2015.03.a	1.0.0	1.5.0	1.0-1

For more information about the Amazon ECS-optimized Amazon Linux AMI, including AMI IDs for the latest version in each Region, see [Amazon ECS-optimized AMI \(p. 333\)](#).

Updating the Amazon ECS container agent

Occasionally, you may need to update the Amazon ECS container agent to pick up bug fixes and new features. Updating the Amazon ECS container agent does not interrupt running tasks or services on the container instance. The process for updating the agent differs depending on whether your container instance was launched with an Amazon ECS-optimized AMI or another operating system.

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

Topics

- [Checking the Amazon ECS container agent version \(p. 448\)](#)
- [Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI \(p. 449\)](#)
- [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 452\)](#)

Checking the Amazon ECS container agent version

You can check the version of the container agent that is running on your container instances to see if you need to update it. The container instance view in the Amazon ECS console provides the agent version. Use the following procedure to check your agent version.

To check the Amazon ECS container agent version (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, select the cluster that hosts the container instance or instances to check.
3. On the **Cluster : *cluster_name*** page, choose **ECS Instances**.

4. Note the **Agent version** column for your container instances. If the container instance does not contain the latest version of the container agent, the console alerts you with a message and flags the outdated agent version.

If your agent version is outdated, you can update your container agent with the following procedures:

- If your container instance is running an Amazon ECS-optimized AMI, see [Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI \(p. 449\)](#).
- If your container instance is not running an Amazon ECS-optimized AMI, see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 452\)](#).

An Amazon ECS-optimized AMI will have "ECS-Optimized" in the AMI name, for example, "Windows_Server-2019-English-Core-ECS_Optimized-2021.10.25". You can view the AMI name on the instance details page in the Amazon EC2 console.

Important

To update the Amazon ECS agent version from versions before v1.0.0 on your Amazon ECS-optimized AMI, we recommend that you terminate your current container instance and launch a new instance with the most recent AMI version. Any container instances that use a preview version should be retired and replaced with the most recent AMI. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

You can also use the Amazon ECS container agent introspection API to check the agent version from the container instance itself. For more information, see [Amazon ECS container agent introspection \(p. 503\)](#).

To check if your Amazon ECS container agent is running the latest version with the introspection API

1. Log in to your container instance via SSH.
2. Query the introspection API.

```
[ec2-user ~]$ curl -s 127.0.0.1:51678/v1/metadata | python -mjson.tool
```

Note

The introspection API added `Version` information in the version v1.0.0 of the Amazon ECS container agent. If `Version` is not present when querying the introspection API, or the introspection API is not present in your agent at all, then the version you are running is v0.0.3 or earlier. You should update your version.

Updating the Amazon ECS container agent on an Amazon ECS-optimized AMI

If you are using an Amazon ECS-optimized AMI, you have several options to get the latest version of the Amazon ECS container agent (shown in order of recommendation):

- Terminate the container instance and launch the latest version of the Amazon ECS-optimized Amazon Linux 2 AMI (either manually or by updating your Auto Scaling launch configuration with the latest AMI). This provides a fresh container instance with the most current tested and validated versions of Amazon Linux, Docker, `ecs-init`, and the Amazon ECS container agent. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).
- Connect to the instance with SSH and update the `ecs-init` package (and its dependencies) to the latest version. This operation provides the most current tested and validated versions of Docker and

`ecs-init` that are available in the Amazon Linux repositories and the latest version of the Amazon ECS container agent. For more information, see [To update the `ecs-init` package on an Amazon ECS-optimized AMI \(p. 450\)](#).

- Update the container agent with the `UpdateContainerAgent` API operation, either through the console or with the AWS CLI or AWS SDKs. For more information, see [Updating the Amazon ECS container agent with the `UpdateContainerAgent` API operation \(p. 450\)](#).

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

To update the `ecs-init` package on an Amazon ECS-optimized AMI

1. Log in to your container instance via SSH. For more information, see [Connect to your container instance \(p. 384\)](#).
2. Update the `ecs-init` package with the following command.

```
sudo yum update -y ecs-init
```

Note

The `ecs-init` package and the Amazon ECS container agent are updated immediately. However, newer versions of Docker are not loaded until the Docker daemon is restarted. Restart either by rebooting the instance, or by running the following commands on your instance:

- Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart docker
```

- Amazon ECS-optimized Amazon Linux AMI:

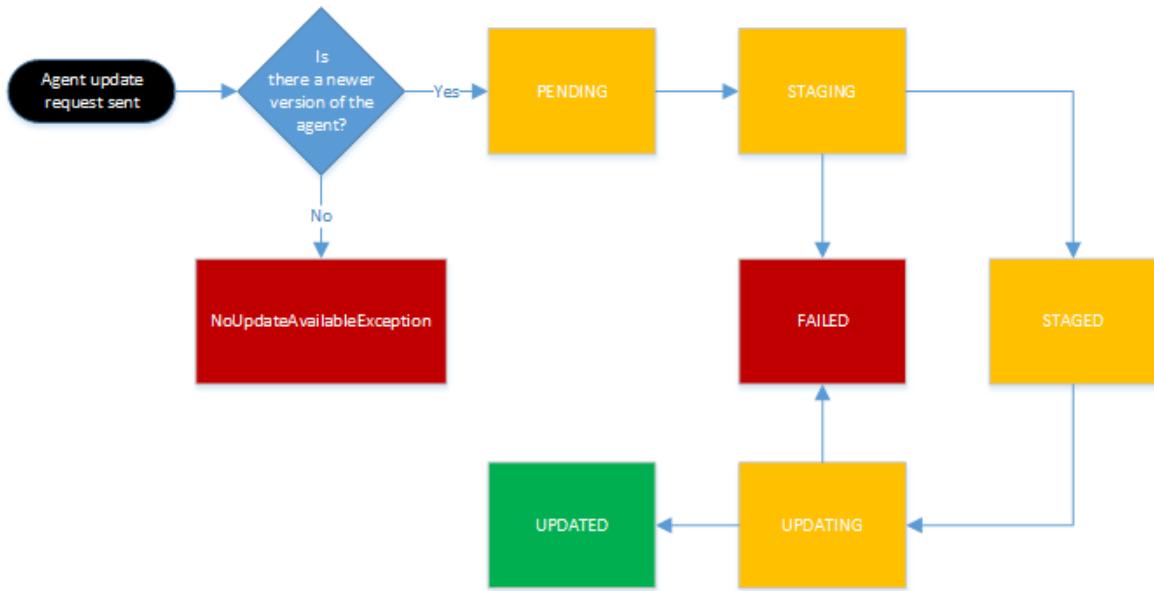
```
sudo service docker restart && sudo start ecs
```

Updating the Amazon ECS container agent with the `UpdateContainerAgent` API operation

Important

The `UpdateContainerAgent` API is only supported on Linux variants of the Amazon ECS-optimized AMI, with the exception of the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI. For container instances using the Amazon ECS-optimized Amazon Linux 2 (arm64) AMI, update the `ecs-init` package to update the agent. For container instances that are running other operating systems, see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 452\)](#). If you are using Windows container instances, we recommend that you launch new container instances to update the agent version in your Windows clusters.

The `UpdateContainerAgent` API process begins when you request an agent update, either through the console or with the AWS CLI or AWS SDKs. Amazon ECS checks your current agent version against the latest available agent version, and if an update is possible, the update process progresses as shown in the flow chart below. If an update is not available, for example, if the agent is already running the most recent version, then a `NoUpdateAvailableException` is returned.



The stages in the update process shown above are as follows:

PENDING

An agent update is available, and the update process has started.

STAGING

The agent has begun downloading the agent update. If the agent cannot download the update, or if the contents of the update are incorrect or corrupted, then the agent sends a notification of the failure and the update transitions to the FAILED state.

STAGED

The agent download has completed and the agent contents have been verified.

UPDATING

The `ecs-init` service is restarted and it picks up the new agent version. If the agent is for some reason unable to restart, the update transitions to the FAILED state; otherwise, the agent signals Amazon ECS that the update is complete.

To update the Amazon ECS container agent on an Amazon ECS-optimized AMI in the console

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, select the cluster that hosts the container instance or instances to check.
3. On the **Cluster : *cluster_name*** page, choose **ECS Instances**.
4. Select the container instance to update.
5. On the **Container Instance** page, choose **Update agent**.

To update the Amazon ECS container agent on an Amazon ECS-optimized AMI with the AWS CLI

Note

Agent updates with the `UpdateContainerAgent` API operation do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

- Use the following command to update the Amazon ECS container agent on your container instance:

```
aws ecs update-container-agent --cluster cluster_name --container-instance container_instance_id
```

Manually updating the Amazon ECS container agent (for non-Amazon ECS-Optimized AMIs)

To manually update the Amazon ECS container agent (for non-Amazon ECS-optimized AMIs)

Note

Agent updates do not apply to Windows container instances. We recommend that you launch new container instances to update the agent version in your Windows clusters.

1. Log in to your container instance via SSH.
2. Check to see if your agent uses the `ECS_DATADIR` environment variable to save its state.

```
ubuntu:~$ docker inspect ecs-agent | grep ECS_DATADIR
```

Output:

```
"ECS_DATADIR=/data",
```

Important

If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before updating your agent. Newer agents with the `ECS_DATADIR` environment variable save their state and you can update them while tasks are running without issues.

3. Stop the Amazon ECS container agent.

```
ubuntu:~$ docker stop ecs-agent
```

4. Delete the agent container.

```
ubuntu:~$ docker rm ecs-agent
```

5. Ensure that the `/etc/ecs` directory and the Amazon ECS container agent configuration file exist at `/etc/ecs/ecs.config`.

```
ubuntu:~$ sudo mkdir -p /etc/ecs && sudo touch /etc/ecs/ecs.config
```

6. Edit the `/etc/ecs/ecs.config` file and ensure that it contains at least the following variable declarations. If you do not want your container instance to register with the default cluster, specify your cluster name as the value for `ECS_CLUSTER`.

```
ECS_DATADIR=/data
ECS_ENABLE_TASK_IAM_ROLE=true
ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true
ECS_LOGFILE=/log/ecs-agent.log
ECS_AVAILABLE_LOGGING_DRIVERS=["json-file","awslogs"]
ECS_LOGLEVEL=info
ECS_CLUSTER=default
```

For more information about these and other agent runtime options, see [Amazon ECS container agent configuration \(p. 454\)](#).

Note

You can optionally store your agent environment variables in Amazon S3 (which can be downloaded to your container instances at launch time using Amazon EC2 user data). This is recommended for sensitive information such as authentication credentials for private repositories. For more information, see [Storing Container Instance Configuration in Amazon S3 \(p. 467\)](#) and [Private registry authentication for tasks \(p. 300\)](#).

7. Pull the latest Amazon ECS container agent image from Docker Hub.

```
ubuntu:~$ docker pull amazon/amazon-ecs-agent:latest
```

Output:

```
Pulling repository amazon/amazon-ecs-agent
a5a56a5e13dc: Download complete
511136ea3c5a: Download complete
9950b5d678a1: Download complete
c48ddcf21b63: Download complete
Status: Image is up to date for amazon/amazon-ecs-agent:latest
```

8. Run the latest Amazon ECS container agent on your container instance.

Note

Use Docker restart policies or a process manager (such as **upstart** or **systemd**) to treat the container agent as a service or a daemon and ensure that it is restarted after exiting. For more information, see [Automatically start containers](#) and [Restart policies](#) in the Docker documentation. The Amazon ECS-optimized AMI uses the `ecs-init` RPM for this purpose, and you can view the [source code for this RPM](#) on GitHub.

The following example of the agent run command is broken into separate lines to show each option. For more information about these and other agent runtime options, see [Amazon ECS container agent configuration \(p. 454\)](#).

Important

Operating systems with SELinux enabled require the `--privileged` option in your `docker run` command. In addition, for SELinux-enabled container instances, we recommend that you add the `:z` option to the `/log` and `/data` volume mounts. However, the host mounts for these volumes must exist before you run the command or you receive a `no such file or directory` error. Take the following action if you experience difficulty running the Amazon ECS agent on an SELinux-enabled container instance:

- Create the host volume mount points on your container instance.

```
ubuntu:~$ sudo mkdir -p /var/log/ecs /var/lib/ecs/data
```

- Add the `--privileged` option to the `docker run` command below.

- Append the `:z` option to the `/log` and `/data` container volume mounts (for example, `--volume=/var/log/ecs/:/log:z`) to the **docker run** command below.

```
ubuntu:~$ sudo docker run --name ecs-agent \
--detach=true \
--restart=on-failure:10 \
--volume=/var/run:/var/run \
--volume=/var/log/ecs/:/log \
--volume=/var/lib/ecs/data:/data \
--volume=/etc/ecs:/etc/ecs \
--net=host \
--env-file=/etc/ecs/ecs.config \
amazon/amazon-ecs-agent:latest
```

Note

If you receive an `Error` response from `daemon: Cannot start container` message, you can delete the failed container with the **sudo docker rm ecs-agent** command and try running the agent again.

Amazon ECS container agent configuration

The Amazon ECS container agent supports a number of configuration options, most of which should be set through environment variables. The following environment variables are available, and all of them are optional.

If your container instance was launched with a Linux variant of the Amazon ECS-optimized AMI, you can set these environment variables in the `/etc/ecs/ecs.config` file and then restart the agent. You can also write these configuration variables to your container instances with Amazon EC2 user data at launch time. For more information, see [Bootstrapping container instances with Amazon EC2 user data \(p. 368\)](#).

If you are manually starting the Amazon ECS container agent (for non Amazon ECS-optimized AMIs), you can use these environment variables in the **docker run** command that you use to start the agent. Use these variables with the syntax `--env=VARIABLE_NAME=VARIABLE_VALUE`. For sensitive information, such as authentication credentials for private repositories, you should store your agent environment variables in a file and pass them all at one time with the `--env-file path_to_env_file` option.

Topics

- [Available Parameters \(p. 454\)](#)
- [Storing Container Instance Configuration in Amazon S3 \(p. 467\)](#)

Available Parameters

The following are the available Amazon ECS container agent configuration parameters. There are undocumented variables that the agent uses internally that may be visible but that are not intended for customer use. For more information, see [Amazon ECS Container Agent](#) on GitHub.

`ECS_CLUSTER`

Example values: `MyCluster`

Default value on Linux: `default`

Default value on Windows: `default`

The cluster that this agent should check into. If this value is undefined, then the default cluster is assumed. If the default cluster does not exist, the Amazon ECS container agent attempts to create it. If a non-default cluster is specified and it does not exist, then registration fails.

ECS_RESERVED_PORTS

Example values: [22, 80, 5000, 8080]

Default value on Linux: [22, 2375, 2376, 51678, 51679, 51680]

Default value on Windows: [53, 135, 139, 445, 2375, 2376, 3389, 5985, 51678, 51679]

An array of ports that should be marked as unavailable for scheduling on this container instance.

ECS_RESERVED_PORTS_UDP

Example values: [53, 123]

Default value on Linux: []

Default value on Windows: []

An array of UDP ports that should be marked as unavailable for scheduling on this container instance.

ECS_ENGINE_AUTH_TYPE

Example values: dockercfg | docker

Default value on Linux: Null

Default value on Windows: Null

Required for private registry authentication. This is the type of authentication data in **ECS_ENGINE_AUTH_DATA**. For more information, see [Authentication formats \(p. 468\)](#).

ECS_ENGINE_AUTH_DATA

Example values:

- `ECS_ENGINE_AUTH_TYPE=dockercfg: {"https://index.docker.io/v1/": {"auth": "zq212MzEXAMPLE7o6T25Dk0i", "email": "email@example.com"}}`
- `ECS_ENGINE_AUTH_TYPE=docker: {"https://index.docker.io/v1/": {"username": "my_name", "password": "my_password", "email": "email@example.com"}}`

Default value on Linux: Null

Default value on Windows: Null

Required for private registry authentication. If `ECS_ENGINE_AUTH_TYPE=dockercfg`, then the `ECS_ENGINE_AUTH_DATA` value should be the contents of a Docker configuration file (`~/.dockercfg` or `~/.docker/config.json`) created by running `docker login`. If `ECS_ENGINE_AUTH_TYPE=docker`, then the `ECS_ENGINE_AUTH_DATA` value should be a JSON representation of the registry server to authenticate against, as well as the authentication parameters required by that registry such as user name, password, and email address for that account. For more information, see [Authentication formats \(p. 468\)](#).

AWS_DEFAULT_REGION

Example values: us-east-1

Default value on Linux: Taken from Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon EC2 instance metadata.

The region to be used in API requests as well as to infer the correct backend host.

AWS_ACCESS_KEY_ID

Example values: AKIAIOSFODNN7EXAMPLE

Default value on Linux: Taken from Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon EC2 instance metadata.

The [access key](#) used by the agent for all calls.

AWS_SECRET_ACCESS_KEY

Example values: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

Default value on Linux: Taken from Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon EC2 instance metadata.

The [secret key](#) used by the agent for all calls.

AWS_SESSION_TOKEN

Default value on Linux: Taken from Amazon EC2 instance metadata.

Default value on Windows: Taken from Amazon EC2 instance metadata.

The [session token](#) used for temporary credentials.

DOCKER_HOST

Example values: unix:///var/run/docker.sock

Default value on Linux: unix:///var/run/docker.sock

Default value on Windows: npipe://./pipe/docker_engine

Used to create a connection to the Docker daemon; behaves similarly to the environment variable as used by the Docker client.

ECS_LOGFILE

Example values: /ecs-agent.log

Default value on Linux: Null

Default value on Windows: Null

The location where agent logs should be written. If you are running the agent via `ecs-init`, which is the default method when using the Amazon ECS-optimized AMI, the in-container path will be `/log` and `ecs-init` mounts that out to `/var/log/ecs/` on the host.

ECS_LOGLEVEL

Example values: crit, error, warn, info, debug

Default value on Linux: info

Default value on Windows: info

The level of detail to log.

ECS_LOGLEVEL_ON_INSTANCE

Example values: `none, crit, error, warn, info, debug`

Default value on Linux: `none`, if `ECS_LOG_DRIVER` is explicitly set to a non-empty value; otherwise the same value as `ECS_LOGLEVEL`

Default value on Windows: `none`, if `ECS_LOG_DRIVER` is explicitly set to a non-empty value; otherwise the same value as `ECS_LOGLEVEL`

Can be used to override `ECS_LOGLEVEL` and set a level of detail that should be logged in the on-instance log file, separate from the level that is logged in the logging driver. If a logging driver is explicitly set, on-instance logs are turned off by default, but can be turned back on with this variable.

ECS_CHECKPOINT

Example values: `true | false`

Default value on Linux: If `ECS_DATADIR` is explicitly set to a non-empty value, then `ECS_CHECKPOINT` is set to `true`; otherwise, it is set to `false`.

Default value on Windows: If `ECS_DATADIR` is explicitly set to a non-empty value, then `ECS_CHECKPOINT` is set to `true`; otherwise, it is set to `false`.

Whether to save the checkpoint state to the location specified with `ECS_DATADIR`.

ECS_DATADIR

Example values: `/data`

Default value on Linux: `/data/`

Default value on Windows: `C:\ProgramData\Amazon\ECS\data`

The name of the persistent data directory on the container that is running the Amazon ECS container agent. The directory is used to save information about the cluster and the agent state.

ECS_UPDATES_ENABLED

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to exit for Amazon ECS agent updates when they are requested.

ECS_DISABLE_METRICS

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `true`

Whether to disable CloudWatch metrics for Amazon ECS. If this value is set to `true`, CloudWatch metrics are not collected.

ECS_POLL_METRICS

Example values: `true | false`

Default value on Linux: `false` (see description below)

Default value on Windows: `false` (see description below)

Whether to poll or stream when gathering CloudWatch metrics for tasks.

In agent versions prior to 1.40.0, the default value was `false`. In agent versions 1.40.0 through 1.42.0, the default value was `true`. In agent versions 1.43.0 and later, the default value is `false`.

Important

Setting `ECS_POLL_METRICS` to `false` will result in high CPU utilization by the agent, dockerd, and containerd when your Amazon EC2 instance is hosting multiple containers.

`ECS_POLLING_METRICS_WAIT_DURATION`

Example values: `30s`

Default value on Linux: `15s`

Default value on Windows: `15s`

Time to wait to poll for new CloudWatch metrics for a task. Only used when `ECS_POLL_METRICS` is `true`.

`ECS_RESERVED_MEMORY`

Example values: `32`

Default value on Linux: `0`

Default value on Windows: `0`

The amount of memory, in MiB, to remove from the pool that is allocated to your tasks. This effectively reserves that memory for critical system processes including the Docker daemon and the Amazon ECS container agent. For example, if you specify `ECS_RESERVED_MEMORY=256`, then the agent registers the total memory minus 256 MiB for that instance, and 256 MiB of the system memory cannot be allocated by Amazon ECS tasks. For more information, see [Container Instance Memory Management \(p. 382\)](#).

`ECS_AVAILABLE_LOGGING_DRIVERS`

Example values: `["awslogs", "fluentd", "gelf", "json-file", "journald", "splunk", "logentries", "syslog"]`

Default value on Linux: `["json-file", "none"]`

Default value on Windows: `["json-file", "none"]`

Note

If you are using ECS init, the default values are `["json-file", "syslog", "awslogs", "none"]`.

The logging drivers available on the container instance. The Amazon ECS container agent running on a container instance must register the logging drivers available on that instance with the `ECS_AVAILABLE_LOGGING_DRIVERS` environment variable before containers placed on that instance can use log configuration options for those drivers in tasks. For information about how to use the awslogs log driver, see [Using the awslogs log driver \(p. 283\)](#). For more information about the different log drivers available for your Docker version and how to configure them, see [Configure logging drivers](#) in the Docker documentation.

`ECS_DISABLE_PRIVILEGED`

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether launching privileged containers is disabled on the container instance. If this value is set to `true`, privileged containers are not permitted.

`ECS_SELINUX_CAPABLE`

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether SELinux is available on the container instance.

`ECS_APPARMOR_CAPABLE`

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether AppArmor is available on the container instance.

`ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`

Example values: `1h` (Valid time units are "ns", "us" (or "`μs`"), "ms", "s", "m", and "h".)

Default value on Linux: `3h`

Default value on Windows: `3h`

Time to wait from when a task is stopped until the Docker container is removed. As this removes the Docker container data, be aware that if this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. The minimum duration is `1m`; any value shorter than 1 minute is ignored.

`ECS_CONTAINER_STOP_TIMEOUT`

Example values: `10m` (Valid time units are "ns", "us" (or "`μs`"), "ms", "s", "m", and "h".)

Default value on Linux: `30s`

Default value on Windows: `30s`

Time to wait from when a task is stopped before its containers are forcefully stopped if they do not exit normally on their own.

`ECS_CONTAINER_START_TIMEOUT`

Example values: `10m` (Valid time units are "ns", "us" (or "`μs`"), "ms", "s", "m", and "h".)

Default value on Linux: `3m`

Default value on Windows: `8m`

Time to wait before giving up on starting a container.

`HTTP_PROXY`

Example values: `10.0.0.131:3128`

Default value on Linux: Null

Default value on Windows: Null

The hostname (or IP address) and port number of an HTTP proxy to use for the Amazon ECS agent to connect to the internet. For example, this proxy will be used if your container instances do not have external network access through an Amazon VPC internet gateway or NAT gateway or instance. If this variable is set, you must also set the `NO_PROXY` variable to filter Amazon EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see [HTTP proxy configuration \(p. 504\)](#).

`NO_PROXY`

Example values:

- Linux: 169.254.169.254,169.254.170.2,/var/run/docker.sock
- Windows: 169.254.169.254,169.254.170.2,\.\pipe\docker_engine

Default value on Linux: Null

Default value on Windows: Null

The HTTP traffic that should not be forwarded to the specified `HTTP_PROXY`. You must specify 169.254.169.254,/var/run/docker.sock to filter Amazon EC2 instance metadata and Docker daemon traffic from the proxy. For more information, see [HTTP proxy configuration \(p. 504\)](#).

`ECS_ENABLE_TASK_IAM_ROLE`

Example values: `true` | `false`

Default value on Linux: `false`

Default value on Windows: `false`

Note

If you are using `ecs init`, the default value is `true`.

Whether IAM roles for tasks should be enabled on the container instance for task containers with the bridge or default network modes. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

`ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST`

Example values: `true` | `false`

Default value on Linux: `false`

Default value on Windows: `false`

Note

If you are using `ecs init`, the default value is `true`.

Whether IAM roles for tasks should be enabled on the container instance for task containers with the host network mode. This variable is only supported on agent versions 1.12.0 and later. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

`ECS_DISABLE_IMAGE_CLEANUP`

Example values: `true`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to disable automated image cleanup for the Amazon ECS agent. For more information, see [Automated task and image cleanup \(p. 471\)](#).

ECS_IMAGE_CLEANUP_INTERVAL

Example values: 30m

Default value on Linux: 30m

Default value on Windows: 30m

The time interval between automated image cleanup cycles. If set to less than 10 minutes, the value is ignored.

ECS_IMAGE_MINIMUM_CLEANUP_AGE

Example values: 30m

Default value on Linux: 1h

Default value on Windows: 1h

The minimum time interval between when an image is pulled and when it can be considered for automated image cleanup.

NON_ECS_IMAGE_MINIMUM_CLEANUP_AGE

Example values: 30m

Default value on Linux: 1h

Default value on Windows: 1h

The minimum time interval between when a non-Amazon ECS image is created and when it can be considered for automated image cleanup.

ECS_NUM_IMAGES_DELETE_PER_CYCLE

Example values: 5

Default value on Linux: 5

Default value on Windows: 5

The maximum number of images to delete in a single automated image cleanup cycle. If set to less than 1, the value is ignored.

ECS_IMAGE_PULL_BEHAVIOR

Example values: default | always | once | prefer-cached

Default value on Linux: default

Default value on Windows: default

The behavior used to customize the pull image process for your container instances. The following describes the optional behaviors:

- If `default` is specified, the image is pulled remotely. If the image pull fails, then the container uses the cached image on the instance.
- If `always` is specified, the image is always pulled remotely. If the image pull fails, then the task fails. This option ensures that the latest version of the image is always pulled. Any cached images are ignored and are subject to the automated image cleanup process.
- If `once` is specified, the image is pulled remotely only if it has not been pulled by a previous task on the same container instance or if the cached image was removed by the automated image

cleanup process. Otherwise, the cached image on the instance is used. This ensures that no unnecessary image pulls are attempted.

- If `prefer-cached` is specified, the image is pulled remotely if there is no cached image. Otherwise, the cached image on the instance is used. Automated image cleanup is disabled for the container to ensure that the cached image is not removed.

`ECS_IMAGE_PULL_INACTIVITY_TIMEOUT`

Example values: `1m`

Default value on Linux: `1m`

Default value on Windows: `3m`

The time to wait after docker pulls complete waiting for extraction of a container. Useful for tuning large Windows containers.

`ECS_INSTANCE_ATTRIBUTES`

Example values: `{ "custom_attribute": "custom_attribute_value" }`

Default value on Linux: Null

Default value on Windows: Null

A list of custom attributes, in JSON format, to apply to your container instances. Using this attribute at instance registration adds the custom attributes, allowing you to skip the manual method of adding custom attributes through the AWS Management Console.

Note

Attributes added do not apply to container instances that are already registered.

To add custom attributes to already-registered container instances, see [Adding an attribute \(p. 518\)](#).

For information about custom attributes to use, see [Attributes \(p. 517\)](#).

An invalid JSON value for this variable causes the agent to exit with a code of 5. A message appears in the agent logs. The JSON value may be valid but there is an issue detected when validating the attribute, such as when the value is too long or contains invalid characters. In that case, the container instance registration happens, but the agent exits with a code of 5 and a message is written to the agent logs. For information about how to locate the agent logs, see [Amazon ECS Container Agent Log \(p. 829\)](#).

`ECS_ENABLE_TASK_ENI`

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to enable task networking for tasks to be launched with their own network interface.

`ECS_CNI_PLUGINS_PATH`

Example values: `/ecs/cni`

Default value on Linux: `/amazon-ecs-cni-plugins`

Default value on Windows: Not applicable

The path where the cni binary file is located on Linux.

ECS_AWSVPC_BLOCK_IMDS

Example values: true | false

Default value on Linux: false

Default value on Windows: false

Whether to block access to [Instance Metadata](#) for tasks started with awsvpc network mode.

ECS_AWSVPC_ADDITIONAL_LOCAL_ROUTES

Example values: ["10.0.15.0/24"]

Default value on Linux: []

Default value on Windows: []

In awsvpc network mode, traffic to these prefixes is routed via the host bridge instead of the task elastic network interface.

ECS_ENABLE_CONTAINER_METADATA

Example values: true | false

Default value on Linux: false

Default value on Windows: false

When true, the agent creates a file describing the container's metadata. The file can be located and consumed by using the container environment variable \$ECS_CONTAINER_METADATA_FILE.

ECS_HOST_DATA_DIR

Example values: /var/lib/ecs

Default value on Linux: /var/lib/ecs

Default value on Windows: Not applicable

The source directory on the host from which ECS_DATADIR is mounted. We use this to determine the source mount path for container metadata files when the Amazon ECS agent is running as a container. We do not use this value in Windows because the Amazon ECS agent does not run as a container.

ECS_ENABLE_TASK_CPU_MEM_LIMIT

Example values: true | false

Default value on Linux: true

Default value on Windows: false

Whether to enable task-level CPU and memory limits.

ECS_CGROUP_PATH

Example values: /sys/fs/cgroup

Default value on Linux: /sys/fs/cgroup

Default value on Windows: Not applicable

The root cgroup path that is expected by the Amazon ECS agent. This is the path that is accessible from the agent mount.

ECS_ENABLE_CPU_UNBOUNDED_WINDOWS_WORKAROUND

Example values: `true | false`

Default value on Linux: Not applicable

Default value on Windows: `false`

When `true`, Amazon ECS allows CPU-unbounded (CPU=0) tasks to run along with CPU-bounded tasks in Windows.

ECS_TASK_METADATA_RPS_LIMIT

Example values: `100, 150`

Default value on Linux: `40, 60`

Default value on Windows: `40, 60`

Comma-separated integer values for steady state and burst throttle limits for the task metadata endpoint.

ECS_SHARED_VOLUME_MATCH_FULL_CONFIG

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

When `dockerVolumeConfiguration` is specified in a task definition and the `autoprovision` flag is used, the Amazon ECS container agent compares the details of the Docker volume with the details of existing Docker volumes. When `ECS_SHARED_VOLUME_MATCH_FULL_CONFIG` is `true`, the container agent compares the full configuration of the volume (`name`, `driverOpts`, and `labels`) to verify that the volumes are identical. When it is `false`, the container agent uses Docker's default behavior, which verifies the volume `name` only. If a volume is shared across container instances, this should be set to `false`. For more information, see [Docker volumes \(p. 263\)](#).

ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM

Example values: `ec2_instance`

Default value on Linux: `none`

Default value on Windows: `none`

If `ec2_instance` is specified, existing tags defined on the container instance are registered to Amazon ECS. The tags are discoverable using the `ListTagsForResource` operation. The IAM role associated with the container instance should have the `ec2:DescribeTags` action allowed. For more information, see [Adding tags to an Amazon EC2 container instance \(p. 608\)](#).

ECS_CONTAINER_INSTANCE_TAGS

Example values: `{ "tag_key": "tag_val" }`

Default value on Linux: `{ }`

Default value on Windows: `{ }`

Metadata applied to container instances to help you categorize and organize your resources. Each tag consists of a custom-defined key and an optional value. Tag keys can have a maximum character length of 128 characters. Tag values can have a maximum length of 256 characters.

If container instance tags are propagated using the `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM` parameter, those tags are overwritten by the tags specified using `ECS_CONTAINER_INSTANCE_TAGS`. For more information, see [Adding tags to an Amazon EC2 container instance \(p. 608\)](#).

`ECS_ENABLE_UNTRACKED_IMAGE_CLEANUP`

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to allow the Amazon ECS agent to delete containers and images that are not part of Amazon ECS tasks.

`ECS_EXCLUDE_UNTRACKED_IMAGE`

Example values: `{"alpine": "latest"}`

Default value on Linux: `{}`

Default value on Windows: `{}`

Comma separated list of images (`imageName:tag`) that should not be deleted by the Amazon ECS agent if `ECS_ENABLE_UNTRACKED_IMAGE_CLEANUP` is `true`.

`ECS_DISABLE_DOCKER_HEALTH_CHECK`

Example values: `true | false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to disable the Docker container health check for the Amazon ECS agent.

`ECS_NVIDIA_RUNTIME`

Example values: `nvidia`

Default value on Linux: `nvidia`

Default value on Windows: `n/a`

The runtime to be used to pass NVIDIA GPU devices to containers. This parameter should not be specified as an environment variable in a task definition if the GPU resource requirements are already specified. For more information, see [Working with GPUs on Amazon ECS \(p. 250\)](#).

`ECS_ENABLE_SPOT_INSTANCE_DRAINING`

Example values: `true`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to enable Spot Instance draining for the container instance. When true, if the container instance receives a Spot interruption notice, then the agent sets the instance status to `DRAINING`, which gracefully shuts down and replaces all tasks running on the instance that are part of a service. It is recommended that this be set to true when using Spot Instances. For more information, see [Container instance draining \(p. 428\)](#).

ECS_LOG_ROLLOVER_TYPE

Example values: `size`, `hourly`

Default value on Linux: `hourly`

Default value on Windows: `hourly`

Determines whether the container agent log file will be rotated hourly or based on size. By default, the agent log file is rotated each hour.

ECS_LOG_OUTPUT_FORMAT

Example values: `logfmt`, `json`

Default value on Linux: `logfmt`

Default value on Windows: `logfmt`

Determines the log output format. When the `json` format is used, each line in the log will be a structured JSON map.

ECS_LOG_MAX_FILE_SIZE_MB

Example values: 10

Default value on Linux: 10

Default value on Windows: 10

When the `ECS_LOG_ROLLOVER_TYPE` variable is set to `size`, this variable determines the maximum size (in MB) of the log file before it is rotated. If the rollover type is set to `hourly`, then this variable is ignored.

ECS_LOG_MAX_ROLL_COUNT

Example values: 24

Default value on Linux: 24

Default value on Windows: 24

Determines the number of rotated log files to keep. Older log files are deleted after this limit is reached.

ECS_ENABLE_AWSLOGS_EXECUTIONROLE_OVERRIDE

Example values: `true`

Default value on Linux: *see description below*

Default value on Windows: `false`

Whether to enable the `awslogs` log driver to authenticate using the task execution IAM role. This parameter must be `true` to use the `awslogs` log driver in a task that also has a task execution IAM role specified. When using the Amazon ECS-optimized AMI with version 1.16.0-1 or later of the `ecs-init` package, the default value of `true` is used. When using older versions of the `ecs-init` package, the default value of `false` is used.

ECS_PULL_DEPENDENT_CONTAINERS_UPFRONT

Example values: `true` | `false`

Default value on Linux: `false`

Default value on Windows: `false`

Whether to pull images for containers with dependencies before the `dependsOn` condition has been satisfied.

`ECS_EXCLUDE_IPV6_PORTBINDING`

Example values: `true | false`

Default value on Linux: `true`

Default value on Windows: `true`

Whether the agent should exclude IPv6 port bindings when the `default` network mode is used. When this value is true, IPv6 port bindings are filtered and task IPv6 port bindings are not returned in the [DescribeTasks](#) response. The bindings are included in the task metadata endpoint.

This is available in agent version 1.55.3 and later.

Storing Container Instance Configuration in Amazon S3

Amazon ECS container agent configuration is controlled with the environment variables described in the previous section. Linux variants of the Amazon ECS-optimized AMI look for these variables in `/etc/ecs/ecs.config` when the container agent starts and configure the agent accordingly. Certain innocuous environment variables, such as `ECS_CLUSTER`, can be passed to the container instance at launch through Amazon EC2 user data and written to this file without consequence. However, other sensitive information, such as your AWS credentials or the `ECS_ENGINE_AUTH_DATA` variable, should never be passed to an instance in user data or written to `/etc/ecs/ecs.config` in a way that would allow them to show up in a `.bash_history` file.

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance configuration at launch. You can store a copy of your `ecs.config` file in a private bucket. You can then use Amazon EC2 user data to install the AWS CLI and copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

To allow Amazon S3 read-only access for your container instance role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and select the IAM role to use for your container instances. This role is likely titled `ecsInstanceRole`. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
3. Under **Managed Policies**, choose **Attach Policy**.
4. To narrow the policy results, on the **Attach Policy** page, for **Filter**, type `s3`.
5. Select the box to the left of the `AmazonS3ReadOnlyAccess` policy and choose **Attach Policy**.

To store an `ecs.config` file in Amazon S3

1. Create an `ecs.config` file with valid environment variables and values from [Amazon ECS container agent configuration \(p. 454\)](#) using the following format. This example configures private registry authentication. For more information, see [Private registry authentication for tasks \(p. 300\)](#).

```
ECS_ENGINE_AUTH_TYPE=dockercfg
```

```
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":  
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

2. To store your configuration file, create a private bucket in Amazon S3. For more information, see [Create a Bucket](#) in the *Amazon Simple Storage Service User Guide*.
3. Upload the `ecs.config` file to your S3 bucket. For more information, see [Add an Object to a Bucket](#) in the *Amazon Simple Storage Service User Guide*.

To load an `ecs.config` file from Amazon S3 at launch

1. Complete the earlier procedures in this section to allow read-only Amazon S3 access to your container instances and store an `ecs.config` file in a private S3 bucket.
2. Launch new container instances by following the steps in [Launching an Amazon ECS Linux container instance \(p. 364\)](#). In [Step 6.g \(p. 365\)](#), use the following example script that installs the AWS CLI and copies your configuration file to `/etc/ecs/ecs.config`.

```
#!/bin/bash  
yum install -y aws-cli  
aws s3 cp s3://your_bucket_name/ecs.config /etc/ecs/ecs.config
```

Private registry authentication for container instances

The Amazon ECS container agent can authenticate with private registries, including Docker Hub, using basic authentication. When you enable private registry authentication, you can use private Docker images in your task definitions. This feature is only supported by tasks using the EC2 launch type.

Another method of enabling private registry authentication uses AWS Secrets Manager to store your private registry credentials securely and then reference them in your container definition. This allows your tasks to use images from private repositories. This method supports tasks using either the EC2 or Fargate launch types. For more information, see [Private registry authentication for tasks \(p. 300\)](#).

The Amazon ECS container agent looks for two environment variables when it launches:

- `ECS_ENGINE_AUTH_TYPE`, which specifies the type of authentication data that is being sent.
- `ECS_ENGINE_AUTH_DATA`, which contains the actual authentication credentials.

Linux variants of the Amazon ECS-optimized AMI scan the `/etc/ecs/ecs.config` file for these variables when the container instance launches, and each time the service is started (with the `sudo start ecs` command). AMIs that are not Amazon ECS-optimized should store these environment variables in a file and pass them with the `--env-file path_to_env_file` option to the `docker run` command that starts the container agent.

Important

We do not recommend that you inject these authentication environment variables at instance launch with Amazon EC2 user data or pass them with the `--env` option to the `docker run` command. These methods are not appropriate for sensitive data, such as authentication credentials. For information about safely adding authentication credentials to your container instances, see [Storing Container Instance Configuration in Amazon S3 \(p. 467\)](#).

Authentication formats

There are two available formats for private registry authentication, `dockercfg` and `docker`.

dockercfg authentication format

The dockercfg format uses the authentication information stored in the configuration file that is created when you run the **docker login** command. You can create this file by running **docker login** on your local system and entering your registry user name, password, and email address. You can also log in to a container instance and run the command there. Depending on your Docker version, this file is saved as either `~/.dockercfg` or `~/.docker/config.json`.

```
cat ~/.docker/config.json
```

Output:

```
{  
  "auths": {  
    "https://index.docker.io/v1/": {  
      "auth": "zq212MzEXAMPLE7o6T25Dk0i"  
    }  
  }  
}
```

Important

Newer versions of Docker create a configuration file as shown above with an outer auths object. The Amazon ECS agent only supports dockercfg authentication data that is in the below format, without the auths object. If you have the **jq** utility installed, you can extract this data with the following command: `cat ~/.docker/config.json | jq .auths`

```
cat ~/.docker/config.json | jq .auths
```

Output:

```
{  
  "https://index.docker.io/v1/": {  
    "auth": "zq212MzEXAMPLE7o6T25Dk0i",  
    "email": "email@example.com"  
  }  
}
```

In the above example, the following environment variables should be added to the environment variable file (`/etc/ecs/ecs.config` for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at runtime. If you are not using an Amazon ECS-optimized AMI and you are starting the agent manually with **docker run**, specify the environment variable file with the `--env-file path_to_env_file` option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=dockercfg  
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":  
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example.com"}}
```

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=dockercfg  
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":  
{"auth":"zq212MzEXAMPLE7o6T25Dk0i","email":"email@example-01.com"}, "repo.example-02.com":  
{"auth":"fQ172MzEXAMPLEof7225DU0j","email":"email@example-02.com"}}
```

docker authentication format

The docker format uses a JSON representation of the registry server that the agent should authenticate with. It also includes the authentication parameters required by that registry (such as user name, password, and the email address for that account). For a Docker Hub account, the JSON representation looks like the following:

```
{  
    "https://index.docker.io/v1/": {  
        "username": "my_name",  
        "password": "my_password",  
        "email": "email@example.com"  
    }  
}
```

In this example, the following environment variables should be added to the environment variable file (/etc/ecs/ecs.config for the Amazon ECS-optimized AMI) that the Amazon ECS container agent loads at runtime. If you are not using an Amazon ECS-optimized AMI, and you are starting the agent manually with `docker run`, specify the environment variable file with the --env-file `path_to_env_file` option when you start the agent.

```
ECS_ENGINE_AUTH_TYPE=docker  
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":  
{"username": "my_name", "password": "my_password", "email": "email@example.com"}}
```

You can configure multiple private registries with the following syntax:

```
ECS_ENGINE_AUTH_TYPE=docker  
ECS_ENGINE_AUTH_DATA={"repo.example-01.com":  
{"username": "my_name", "password": "my_password", "email": "email@example-01.com"}, "repo.example-02.com":  
{"username": "another_name", "password": "another_password", "email": "email@example-02.com"}}
```

Enabling private registries

Use the following procedure to enable private registries for your container instances.

To enable private registries in the Amazon ECS-optimized AMI

1. Log in to your container instance using SSH.
2. Open the /etc/ecs/ecs.config file and add the `ECS_ENGINE_AUTH_TYPE` and `ECS_ENGINE_AUTH_DATA` values for your registry and account:

```
sudo vi /etc/ecs/ecs.config
```

This example authenticates a Docker Hub user account:

```
ECS_ENGINE_AUTH_TYPE=docker  
ECS_ENGINE_AUTH_DATA={"https://index.docker.io/v1/":  
{"username": "my_name", "password": "my_password", "email": "email@example.com"}}
```

3. Check to see if your agent uses the `ECS_DATADIR` environment variable to save its state:

```
docker inspect ecs-agent | grep ECS_DATADIR
```

Output:

```
"ECS_DATADIR=/data",
```

Important

If the previous command does not return the `ECS_DATADIR` environment variable, you must stop any tasks running on this container instance before stopping the agent. Newer agents with the `ECS_DATADIR` environment variable save their state and you can stop and start them while tasks are running without issues. For more information, see [Updating the Amazon ECS container agent \(p. 448\)](#).

4. Stop the `ecs` service:

```
sudo stop ecs
```

Output:

```
ecs stop/waiting
```

5. Restart the `ecs` service.

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI:

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

6. (Optional) You can verify that the agent is running and see some information about your new container instance by querying the agent introspection API operation. For more information, see [the section called "Container agent introspection" \(p. 503\)](#).

```
curl http://localhost:51678/v1/metadata
```

Automated task and image cleanup

Each time a task is placed on a container instance, the Amazon ECS container agent checks to see if the images referenced in the task are the most recent of the specified tag in the repository. If not, the default behavior allows the agent to pull the images from their respective repositories. If you frequently update the images in your tasks and services, your container instance storage can quickly fill up with Docker images that you are no longer using and may never use again. For example, you may use a continuous integration and continuous deployment (CI/CD) pipeline.

Note

The Amazon ECS agent image pull behavior can be customized using the `ECS_IMAGE_PULL_BEHAVIOR` parameter. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

Likewise, containers that belong to stopped tasks can also consume container instance storage with log information, data volumes, and other artifacts. These artifacts are useful for debugging containers that have stopped unexpectedly, but most of this storage can be safely freed up after a period of time.

By default, the Amazon ECS container agent automatically cleans up stopped tasks and Docker images that are not being used by any tasks on your container instances.

Note

The automated image cleanup feature requires at least version 1.13.0 of the Amazon ECS container agent. To update your agent to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

Tunable parameters

The following agent configuration variables are available to tune your automated task and image cleanup experience. For more information about how to set these variables on your container instances, see [Amazon ECS container agent configuration \(p. 454\)](#).

`ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`

This variable specifies the time to wait before removing any containers that belong to stopped tasks. The image cleanup process cannot delete an image as long as there is a container that references it. After images are not referenced by any containers (either stopped or running), then the image becomes a candidate for cleanup. By default, this parameter is set to 3 hours but you can reduce this period to as low as 1 minute, if you need to for your application.

`ECS_DISABLE_IMAGE_CLEANUP`

If you set this variable to `true`, then automated image cleanup is disabled on your container instance and no images are automatically removed.

`ECS_IMAGE_CLEANUP_INTERVAL`

This variable specifies how frequently the automated image cleanup process should check for images to delete. The default is every 30 minutes but you can reduce this period to as low as 10 minutes to remove images more frequently.

`ECS_IMAGE_MINIMUM_CLEANUP_AGE`

This variable specifies the minimum amount of time between when an image was pulled and when it may become a candidate for removal. This is used to prevent cleaning up images that have just been pulled. The default is 1 hour.

`ECS_NUM_IMAGES_DELETE_PER_CYCLE`

This variable specifies how many images may be removed during a single cleanup cycle. The default is 5 and the minimum is 1.

Cleanup workflow

When the Amazon ECS container agent is running and automated image cleanup is not disabled, the agent checks for Docker images that are not referenced by running or stopped containers at a frequency determined by the `ECS_IMAGE_CLEANUP_INTERVAL` variable. If unused images are found and they are older than the minimum cleanup time specified by the `ECS_IMAGE_MINIMUM_CLEANUP_AGE` variable, the agent removes up to the maximum number of images that are specified with the `ECS_NUM_IMAGES_DELETE_PER_CYCLE` variable. The least-recently referenced images are deleted first. After the images are removed, the agent waits until the next interval and repeats the process again.

Amazon ECS container metadata file

Beginning with version 1.15.0 of the Amazon ECS container agent, various container metadata is available within your containers or the host container instance. By enabling this feature, you can query the information about a task, container, and container instance from within the container or the host container instance. The metadata file is created on the host instance and mounted in the container as a Docker volume and therefore is not available when a task is hosted on AWS Fargate.

The container metadata file is cleaned up on the host instance when the container is cleaned up. You can adjust when this happens with the `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION` container agent variable. For more information, see [Automated task and image cleanup \(p. 471\)](#).

Topics

- [Enabling container metadata \(p. 473\)](#)
- [Container metadata file locations \(p. 473\)](#)
- [Container metadata file format \(p. 474\)](#)

Enabling container metadata

This feature is disabled by default. You can enable container metadata at the container instance level by setting the `ECS_ENABLE_CONTAINER_METADATA` container agent variable to `true`. You can set this variable in the `/etc/ecs/ecs.config` configuration file and restart the agent. You can also set it as a Docker environment variable at runtime when the agent container is started. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

If the `ECS_ENABLE_CONTAINER_METADATA` is set to `true` when the agent starts, metadata files are created for any containers created from that point forward. The Amazon ECS container agent cannot create metadata files for containers that were created before the `ECS_ENABLE_CONTAINER_METADATA` container agent variable was set to `true`. To ensure that all containers receive metadata files, you should set this agent variable at container instance launch. The following is an example user data script that will set this variable as well as register your container instance with your cluster.

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=your_cluster_name
ECS_ENABLE_CONTAINER_METADATA=true
EOF
```

Container metadata file locations

By default, the container metadata file is written to the following host and container paths.

- **For Linux instances:**

- Host path: `/var/lib/ecs/data/metadata/cluster_name/task_id/container_name/ecs-container-metadata.json`

Note

The Linux host path assumes that the default data directory mount path (`/var/lib/ecs/data`) is used when the agent is started. If you are not using an Amazon ECS-optimized AMI (or the `ecs-init` package to start and maintain the container agent), be sure to set the `ECS_HOST_DATA_DIR` agent configuration variable to the host path where the container agent's state file is located. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

- Container path: `/opt/ecs/metadata/random_ID/ecs-container-metadata.json`

- **For Windows instances:**

- Host path: `C:\ProgramData\Amazon\ECS\data\metadata\task_id\container_name\ecs-container-metadata.json`
- Container path: `C:\ProgramData\Amazon\ECS\metadata\random_ID\ecs-container-metadata.json`

However, for easy access, the container metadata file location is set to the `ECS_CONTAINER_METADATA_FILE` environment variable inside the container. You can read the file contents from inside the container with the following command:

- **For Linux instances:**

```
cat $ECS_CONTAINER_METADATA_FILE
```

- **For Windows instances (PowerShell):**

```
Get-Content -path $env:ECS_CONTAINER_METADATA_FILE
```

Container metadata file format

The following information is stored in the container metadata JSON file.

Cluster

The name of the cluster that the container's task is running on.

ContainerInstanceARN

The full Amazon Resource Name (ARN) of the host container instance.

TaskARN

The full Amazon Resource Name (ARN) of the task that the container belongs to.

TaskDefinitionFamily

The name of the task definition family the container is using.

TaskDefinitionRevision

The task definition revision the container is using.

ContainerID

The Docker container ID (and not the Amazon ECS container ID) for the container.

ContainerName

The container name from the Amazon ECS task definition for the container.

DockerContainerName

The container name that the Docker daemon uses for the container (for example, the name that shows up in `docker ps` command output).

ImageID

The SHA digest for the Docker image used to start the container.

ImageName

The image name and tag for the Docker image used to start the container.

PortMappings

Any port mappings associated with the container.

ContainerPort

The port on the container that is exposed.

HostPort

The port on the host container instance that is exposed.

BindIp

The bind IP address that is assigned to the container by Docker. This IP address is only applied with the bridge network mode, and it is only accessible from the container instance.

Protocol

The network protocol used for the port mapping.

Networks

The network mode and IP address for the container.

NetworkMode

The network mode for the task to which the container belongs.

IPv4Addresses

The IP addresses associated with the container.

Important

If your task is using the `awsvpc` network mode, the IP address of the container will not be returned. In this case, you can retrieve the IP address by reading the `/etc/hosts` file with the following command:

```
tail -1 /etc/hosts | awk '{print $1}'
```

MetadataFileStatus

The status of the metadata file. When the status is `READY`, the metadata file is current and complete. If the file is not ready yet (for example, the moment the task is started), a truncated version of the file format is available. To avoid a likely race condition where the container has started, but the metadata has not yet been written, you can parse the metadata file and wait for this parameter to be set to `READY` before depending on the metadata. This is usually available in less than 1 second from when the container starts.

AvailabilityZone

The Availability Zone the host container instance resides in.

HostPrivateIPv4Address

The private IP address for the task the container belongs to.

HostPublicIPv4Address

The public IP address for the task the container belongs to.

Example Amazon ECS container metadata file (`READY`)

The following example shows a container metadata file in the `READY` status.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/2b88376d-aba3-4950-9ddf-bcb0f388a40c",
    "TaskDefinitionFamily": "console-sample-app-static",
    "TaskDefinitionRevision": "1",
    "ContainerID": "aec2557997f4eed9b280c2efd7afcccdcedfda4ac399f7480cae870cf7e163fd",
    "ContainerName": "simple-app",
    "DockerContainerName": "/ecs-console-sample-app-static-1-simple-app-e4e8e495e8baa5de1a00",
    "ImageID": "sha256:2ae34abc2ed0a22e280d17e13f9c01aa725688b09b7a1525d1a2750e2c0d1de",
    "ImageName": "httpd:2.4",
    "PortMappings": [
```

```
{
    "ContainerPort": 80,
    "HostPort": 80,
    "BindIp": "0.0.0.0",
    "Protocol": "tcp"
}
],
"Networks": [
{
    "NetworkMode": "bridge",
    "IPv4Addresses": [
        "192.0.2.0"
    ]
}
],
"MetadataFileStatus": "READY",
"AvailabilityZone": "us-east-1b",
"HostPrivateIPv4Address": "192.0.2.0",
"HostPublicIPv4Address": "203.0.113.0"
}
```

Example Incomplete Amazon ECS container metadata file (not yet READY)

The following example shows a container metadata file that has not yet reached the `READY` status. The information in the file is limited to a few parameters that are known from the task definition. The container metadata file should be ready within 1 second after the container starts.

```
{
    "Cluster": "default",
    "ContainerInstanceARN": "arn:aws:ecs:us-west-2:012345678910:container-instance/
default/1f73d099-b914-411c-a9ff-81633b7741dd",
    "TaskARN": "arn:aws:ecs:us-west-2:012345678910:task/default/
d90675f8-1a98-444b-805b-3d9cab6fc4",
    "ContainerName": "metadata"
}
```

Amazon ECS task metadata endpoint

Important

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Amazon ECS task metadata endpoint](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

The Amazon ECS container agent provides a method to retrieve various task metadata and Docker stats. This is referred to as the task metadata endpoint. The following versions are available:

- Task metadata endpoint version 4 – Provides a variety of metadata and Docker stats to containers. Can also provide network rate data. Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version 1.39.0 of the Amazon ECS container agent. For Amazon EC2 Windows instances that use awsvpc network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Task metadata endpoint version 4 \(p. 477\)](#).
- Task metadata endpoint version 3 – Provides a variety of metadata and Docker stats to containers. Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version 1.21.0 of the Amazon ECS container agent. For Amazon EC2 Windows instances that use awsvpc network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Task Metadata Endpoint version 3 \(p. 493\)](#).
- Task metadata endpoint version 2 – Available for Amazon ECS tasks launched on Amazon EC2 Linux instances running at least version 1.17.0 of the Amazon ECS container agent. For Amazon EC2

Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Task Metadata Endpoint version 2 \(p. 498\)](#).

Task metadata endpoint version 4

Important

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Task metadata endpoint version 4 in the Amazon Elastic Container Service User Guide for AWS Fargate](#).

The Amazon ECS container agent injects an environment variable into each container, referred to as the *task metadata endpoint* which provides various task metadata and `Docker stats` to the container.

The task metadata and network rate stats are sent to CloudWatch Container Insights and can be viewed in the AWS Management Console. For more information, see [Amazon ECS CloudWatch Container Insights \(p. 645\)](#).

Note

Amazon ECS provides earlier versions of the task metadata endpoint. To avoid the need to create new task metadata endpoint versions in the future, additional metadata may be added to the version 4 output. We will not remove any existing metadata or change the metadata field names.

Enabling the task metadata endpoint

The environment variable is injected by default into the containers of Amazon ECS tasks launched on Amazon EC2 Linux instances that are running at least version 1.39.0 of the Amazon ECS container agent. For Amazon EC2 Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version 1.54.0. For more information, see [Amazon ECS container agent versions \(p. 440\)](#).

Note

You can add support for this feature on Amazon EC2 instances using older versions of the Amazon ECS container agent by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent \(p. 448\)](#).

Task metadata endpoint version 4 paths

The following task metadata endpoint paths are available to containers.

`#{ECS_CONTAINER_METADATA_URI_V4}`

This path returns metadata for the container.

`#{ECS_CONTAINER_METADATA_URI_V4}/task`

This path returns metadata for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see [Task metadata JSON response \(p. 478\)](#).

`#{ECS_CONTAINER_METADATA_URI_V4}/taskWithTags`

This path returns the metadata for the task included in the /task endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API. Any errors received when retrieving the tag metadata will be included in the `Errors` field in the response.

Note

The `Errors` field is only in the response for tasks hosted on Amazon EC2 Linux instances running at least version 1.50.0 of the container agent. For Amazon EC2 Windows instances that use `awsvpc` network mode, the Amazon ECS container agent must be at least version 1.54.0.

`${ECS_CONTAINER_METADATA_URI_V4}/stats`

This path returns Docker stats for the specific container. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

For Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 Linux instances running at least version `1.43.0` of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

`${ECS_CONTAINER_METADATA_URI_V4}/task/stats`

This path returns Docker stats for all of the containers associated with the task. This can be used by sidecar containers to extract network metrics. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

For Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 Linux instances running at least version `1.43.0` of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

Task metadata JSON response

The following information is returned from the task metadata endpoint (`${ECS_CONTAINER_METADATA_URI_V4}/task`) JSON response. This includes metadata associated with the task in addition to the metadata for each container within the task.

Cluster

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

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 (Windows).

LaunchType

The launch type the task is using. When using cluster capacity providers, this indicates whether the task is using Fargate or EC2 infrastructure.

Note

This `LaunchType` metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later (Linux) or 1.0.0 or later (Windows).

Containers

A list of container metadata for each container associated with the task.

DockerId

The Docker ID for the container.

When you use Fargate, the id is a 32-digit hex followed by a 10 digit number.

Name

The name of the container as specified in the task definition.

DockerName

The name of the container supplied to Docker. The Amazon ECS container agent generates a unique name for the container to avoid name collisions when multiple copies of the same task definition are run on a single instance.

Image

The image for the container.

ImageID

The SHA-256 digest 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.

LogDriver

The log driver the container is using.

Note

This LogDriver metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later.

LogOptions

The log driver options defined for the container.

Note

This LogOptions metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later.

ContainerARN

The full Amazon Resource Name (ARN) of the container.

Note

This ContainerARN metadata is only included when using Amazon ECS Linux container agent version 1.45.0 or later.

Networks

The network information for the container, such as the network mode and IP address. This parameter is omitted if no network information is defined.

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 example outputs from each of the task metadata endpoints.

Example container metadata response

When querying the `#{ECS_CONTAINER_METADATA_URI_V4}` endpoint you are returned only metadata about the container itself. The following is an example output.

```
{
    "DockerId": "ea32192c8553fbff06c9340478a2ff089b2bb5646fb718b4ee206641c9086d66",
    "Name": "curl",
    "DockerName": "ecs-curltest-24-curl-cca48e8dcadd97805600",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/8f03e41243824aea923aca126495f665",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "24"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:15:07.620912337Z",
    "StartedAt": "2020-10-02T00:15:08.062559351Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/8f03e41243824aea923aca126495f665"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/0206b271-b33f-47ab-86c6-a0ba208a70a9",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.100"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:9e:32:c7:48:85",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-100.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}
```

Example task metadata response

When querying the `#{ECS_CONTAINER_METADATA_URI_V4}/task` endpoint you are returned metadata about the task the container is part of in addition to the metadata for each container within the task. The following is an example output.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
```

```

"AvailabilityZone": "us-west-2d",
"LaunchType": "EC2",
"Containers": [
    {
        "DockerId": "598cba581fe3f939459eabale071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
        "Name": "~internal-ecs-pause",
        "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
        "Image": "amazon/amazon-ecs-pause:0.1.0",
        "ImageID": "",
        "Labels": {
            "com.amazonaws.ecs.cluster": "default",
            "com.amazonaws.ecs.container-name": "~internal-ecs-pause",
            "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
            "com.amazonaws.ecs.task-definition-family": "curltest",
            "com.amazonaws.ecs.task-definition-version": "26"
        },
        "DesiredStatus": "RESOURCES_PROVISIONED",
        "KnownStatus": "RESOURCES_PROVISIONED",
        "Limits": {
            "CPU": 0,
            "Memory": 0
        },
        "CreatedAt": "2020-10-02T00:43:05.602352471Z",
        "StartedAt": "2020-10-02T00:43:06.076707576Z",
        "Type": "CNI_PAUSED",
        "Networks": [
            {
                "NetworkMode": "awsvpc",
                "IPv4Addresses": [
                    "10.0.2.61"
                ],
                "AttachmentIndex": 0,
                "MACAddress": "0e:10:e2:01:bd:91",
                "IPv4SubnetCIDRBlock": "10.0.2.0/24",
                "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
                "SubnetGatewayIpv4Address": "10.0.2.1/24"
            }
        ]
    },
    {
        "DockerId": "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
        "Name": "curl",
        "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
        "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
        "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
        "Labels": {
            "com.amazonaws.ecs.cluster": "default",
            "com.amazonaws.ecs.container-name": "curl",
            "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
            "com.amazonaws.ecs.task-definition-family": "curltest",
            "com.amazonaws.ecs.task-definition-version": "26"
        },
        "DesiredStatus": "RUNNING",
        "KnownStatus": "RUNNING",
        "Limits": {
            "CPU": 10,
            "Memory": 128
        },
        "CreatedAt": "2020-10-02T00:43:06.326590752Z",
        "StartedAt": "2020-10-02T00:43:06.767535449Z",
        "Type": "NORMAL",
        "LogDriver": "awslogs",
        "LogOptions": {

```

```

        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/
abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}

```

Example task with tags metadata response

When querying the `/${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags` endpoint you are returned metadata about the task, including the task and container instance tags. The following is an example output.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "TaskTags": {
        "tag-use": "task-metadata-endpoint-test"
    },
    "ContainerInstanceTags": {
        "tag_key": "tag_value"
    },
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId": "598cba581fe3f939459eabae071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal-ecs-pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal-ecs-pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/
default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",
                "com.amazonaws.ecs.task-definition-version": "26"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",

```

```

    "KnownStatus": "RESOURCES_PROVISIONED",
    "Limits": {
        "CPU": 0,
        "Memory": 0
    },
    "CreatedAt": "2020-10-02T00:43:05.602352471Z",
    "StartedAt": "2020-10-02T00:43:06.076707576Z",
    "Type": "CNI_PAUSE",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
},
{
    "DockerId": "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
    "Name": "curl",
    "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:43:06.326590752Z",
    "StartedAt": "2020-10-02T00:43:06.767535449Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
}

```

```

        }
    ]
}
}
```

Example task with tags with an error metadata response

When querying the `/${ECS_CONTAINER_METADATA_URI_V4}/taskWithTags` endpoint you are returned metadata about the task, including the task and container instance tags. If there is an error retrieving the tagging data, the error is returned in the response. The following is an example output for when the IAM role associated with the container instance doesn't have the `ecs>ListTagsForResource` permission allowed.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
    "Family": "curltest",
    "Revision": "26",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "PullStartedAt": "2020-10-02T00:43:06.202617438Z",
    "PullStoppedAt": "2020-10-02T00:43:06.31288465Z",
    "AvailabilityZone": "us-west-2d",
    "Errors": [
        {
            "ErrorField": "ContainerInstanceTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform: ecs>ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:container-instance/default/2dd1b186f39845a584488d2ef155c131",
            "StatusCode": 400,
            "RequestId": "cd597ef0-272b-4643-9bd2-1de469870fa6",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:container-instance/default/2dd1b186f39845a584488d2ef155c131"
        },
        {
            "ErrorField": "TaskTags",
            "ErrorCode": "AccessDeniedException",
            "ErrorMessage": "User: arn:aws:sts::111122223333:assumed-role/ecsInstanceRole/i-0744a608689EXAMPLE is not authorized to perform: ecs>ListTagsForResource on resource: arn:aws:ecs:us-west-2:111122223333:task/default/9ef30e4b7aa44d0db562749cff4983f3",
            "StatusCode": 400,
            "RequestId": "862c5986-6cd2-4aa6-87cc-70be395531e1",
            "ResourceARN": "arn:aws:ecs:us-west-2:111122223333:task/default/9ef30e4b7aa44d0db562749cff4983f3"
        }
    ],
    "LaunchType": "EC2",
    "Containers": [
        {
            "DockerId": "598cba581fe3f939459eabale071d5c93bb2c49b7d1ba7db6bb19deeb70d8e38",
            "Name": "~internal-ecs-pause",
            "DockerName": "ecs-curltest-26-internalecspause-e292d586b6f9dade4a00",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal-ecs-pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
                "com.amazonaws.ecs.task-definition-family": "curltest",

```

```

        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RESOURCES_PROVISIONED",
    "KnownStatus": "RESOURCES_PROVISIONED",
    "Limits": {
        "CPU": 0,
        "Memory": 0
    },
    "CreatedAt": "2020-10-02T00:43:05.602352471Z",
    "StartedAt": "2020-10-02T00:43:06.076707576Z",
    "Type": "CNI_PAUSE",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
            "IPv4SubnetCIDRBlock": "10.0.2.0/24",
            "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
            "SubnetGatewayIpv4Address": "10.0.2.1/24"
        }
    ]
},
{
    "DockerId": "ee08638adaaf009d78c248913f629e38299471d45fe7dc944d1039077e3424ca",
    "Name": "curl",
    "DockerName": "ecs-curltest-26-curl-a0e7dba5aca6d8cb2e00",
    "Image": "111122223333.dkr.ecr.us-west-2.amazonaws.com/curltest:latest",
    "ImageID": "sha256:d691691e9652791a60114e67b365688d20d19940dde7c4736ea30e660d8d3553",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-west-2:111122223333:task/default/158d1c8083dd49d6b527399fd6414f5c",
        "com.amazonaws.ecs.task-definition-family": "curltest",
        "com.amazonaws.ecs.task-definition-version": "26"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 10,
        "Memory": 128
    },
    "CreatedAt": "2020-10-02T00:43:06.326590752Z",
    "StartedAt": "2020-10-02T00:43:06.767535449Z",
    "Type": "NORMAL",
    "LogDriver": "awslogs",
    "LogOptions": {
        "awslogs-create-group": "true",
        "awslogs-group": "/ecs/metadata",
        "awslogs-region": "us-west-2",
        "awslogs-stream": "ecs/curl/158d1c8083dd49d6b527399fd6414f5c"
    },
    "ContainerARN": "arn:aws:ecs:us-west-2:111122223333:container/abb51bdd-11b4-467f-8f6c-adcfe1fe059d",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.61"
            ],
            "AttachmentIndex": 0,
            "MACAddress": "0e:10:e2:01:bd:91",
        }
    ]
}

```

```

        "IPv4SubnetCIDRBlock": "10.0.2.0/24",
        "PrivateDNSName": "ip-10-0-2-61.us-west-2.compute.internal",
        "SubnetGatewayIpv4Address": "10.0.2.1/24"
    }
}
]
}
}

```

Example container stats response

When querying the `/${ECS_CONTAINER_METADATA_URI_V4}/stats` endpoint you are returned network metrics for the container. For Amazon ECS tasks that use the `awsvpc` or `bridge` network modes hosted on Amazon EC2 instances running at least version `1.43.0` of the container agent, there will be additional network rate stats included in the response. For all other tasks, the response will only include the cumulative network stats.

The following is an example output from an Amazon ECS task on Amazon EC2 that uses the `bridge` network mode.

```
{
  "read": "2020-10-02T00:51:13.410254284Z",
  "preread": "2020-10-02T00:51:12.406202398Z",
  "pids_stats": {
    "current": 3
  },
  "blkio_stats": {
    "io_service_bytes_recursive": [
      ],
      "io_serviced_recursive": [
        ],
        "io_queue_recursive": [
          ],
          "io_service_time_recursive": [
            ],
            "io_wait_time_recursive": [
              ],
              "io_merged_recursive": [
                ],
                "io_time_recursive": [
                  ],
                  "sectors_recursive": [
                    ]
      },
      "num_procs": 0,
      "storage_stats": {

      },
      "cpu_stats": {
        "cpu_usage": {
          "total_usage": 360968065,
          "percpu_usage": [
            182359190,
            178608875
          ],
          "cpu_percent": 100
        }
      }
}
```

```
        "usage_in_kernelmode": 40000000,
        "usage_in_usermode": 290000000
    },
    "system_cpu_usage": 13939680000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"precpu_stats": {
    "cpu_usage": {
        "total_usage": 360968065,
        "percpu_usage": [
            182359190,
            178608875
        ],
        "usage_in_kernelmode": 40000000,
        "usage_in_usermode": 290000000
    },
    "system_cpu_usage": 13937670000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"memory_stats": {
    "usage": 1806336,
    "max_usage": 6299648,
    "stats": {
        "active_anon": 606208,
        "active_file": 0,
        "cache": 0,
        "dirty": 0,
        "hierarchical_memory_limit": 134217728,
        "hierarchical_memsw_limit": 268435456,
        "inactive_anon": 0,
        "inactive_file": 0,
        "mapped_file": 0,
        "pgfault": 4185,
        "pgmajfault": 0,
        "ppgpin": 2926,
        "ppgout": 2778,
        "rss": 606208,
        "rss_huge": 0,
        "total_active_anon": 606208,
        "total_active_file": 0,
        "total_cache": 0,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 0,
        "total_mapped_file": 0,
        "total_pgfault": 4185,
        "total_pgmajfault": 0,
        "total_ppgpin": 2926,
        "total_ppgout": 2778,
        "total_rss": 606208,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
    }
},
```

```

        "limit": 134217728
    },
    "name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
    "id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
    "networks": {
        "eth0": {
            "rx_bytes": 84,
            "rx_packets": 2,
            "rx_errors": 0,
            "rx_dropped": 0,
            "tx_bytes": 84,
            "tx_packets": 2,
            "tx_errors": 0,
            "tx_dropped": 0
        }
    },
    "network_rate_stats": {
        "rx_bytes_per_sec": 0,
        "tx_bytes_per_sec": 0
    }
}

```

Example task stats response

When querying the `/${ECS_CONTAINER_METADATA_URI_V4}/task/stats` endpoint you are returned network metrics about the task the container is part of. The following is an example output.

```
{
    "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854": {
        "read": "2020-10-02T00:51:32.51467703Z",
        "preread": "2020-10-02T00:51:31.50860463Z",
        "pids_stats": {
            "current": 1
        },
        "blkio_stats": {
            "io_service_bytes_recursive": [
                ],
            "io_serviced_recursive": [
                ],
            "io_queue_recursive": [
                ],
            "io_service_time_recursive": [
                ],
            "io_wait_time_recursive": [
                ],
            "io_merged_recursive": [
                ],
            "io_time_recursive": [
                ],
            "sectors_recursive": [
                ]
        },
        "num_procs": 0,
        "storage_stats": {
            }
    }
}
```

```
"cpu_stats": {
    "cpu_usage": {
        "total_usage": 177232665,
        "percpu_usage": [
            13376224,
            163856441
        ],
        "usage_in_kernelmode": 0,
        "usage_in_usermode": 160000000
    },
    "system_cpu_usage": 13977820000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"precpu_stats": {
    "cpu_usage": {
        "total_usage": 177232665,
        "percpu_usage": [
            13376224,
            163856441
        ],
        "usage_in_kernelmode": 0,
        "usage_in_usermode": 160000000
    },
    "system_cpu_usage": 13975800000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"memory_stats": {
    "usage": 532480,
    "max_usage": 6279168,
    "stats": {
        "active_anon": 40960,
        "active_file": 0,
        "cache": 0,
        "dirty": 0,
        "hierarchical_memory_limit": 9223372036854771712,
        "hierarchical_mems_w_limit": 9223372036854771712,
        "inactive_anon": 0,
        "inactive_file": 0,
        "mapped_file": 0,
        "pgfault": 2033,
        "pgmajfault": 0,
        "pgpgin": 1734,
        "pgpgout": 1724,
        "rss": 40960,
        "rss_huge": 0,
        "total_active_anon": 40960,
        "total_active_file": 0,
        "total_cache": 0,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 0,
        "total_mapped_file": 0,
        "total_pgfault": 2033,
        "total_pgmajfault": 0,
        "total_pgpgin": 1734,
        "total_pgpgout": 1724,
    }
}
```

```

        "total_rss": 40960,
        "total_rss_huge": 0,
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
    },
    "limit": 4073377792
},
"name": "/ecs-curltest-26-internalecspause-a6bcc3dbadfacfe85300",
"id": "01999f2e5c6cf4df3873f28950e6278813408f281c54778efec860d0caad4854",
"networks": {
    "eth0": {
        "rx_bytes": 84,
        "rx_packets": 2,
        "rx_errors": 0,
        "rx_dropped": 0,
        "tx_bytes": 84,
        "tx_packets": 2,
        "tx_errors": 0,
        "tx_dropped": 0
    }
},
"network_rate_stats": {
    "rx_bytes_per_sec": 0,
    "tx_bytes_per_sec": 0
}
},
"5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af": {
    "read": "2020-10-02T00:51:32.512771349Z",
    "preread": "2020-10-02T00:51:31.510597736Z",
    "pids_stats": {
        "current": 3
    },
    "blkio_stats": {
        "io_service_bytes_recursive": [
            ],
        "io_serviced_recursive": [
            ],
        "io_queue_recursive": [
            ],
        "io_service_time_recursive": [
            ],
        "io_wait_time_recursive": [
            ],
        "io_merged_recursive": [
            ],
        "io_time_recursive": [
            ],
        "sectors_recursive": [
            ]
    },
    "num_procs": 0,
    "storage_stats": {
        },
    "cpu_stats": {
        "cpu_usage": {
            }
        }
    }
}

```

```
        "total_usage": 379075681,
        "percpu_usage": [
            191355275,
            187720406
        ],
        "usage_in_kernelmode": 60000000,
        "usage_in_usermode": 310000000
    },
    "system_cpu_usage": 13977800000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"precpu_stats": {
    "cpu_usage": {
        "total_usage": 378825197,
        "percpu_usage": [
            191104791,
            187720406
        ],
        "usage_in_kernelmode": 60000000,
        "usage_in_usermode": 310000000
    },
    "system_cpu_usage": 13975800000000,
    "online_cpus": 2,
    "throttling_data": {
        "periods": 0,
        "throttled_periods": 0,
        "throttled_time": 0
    }
},
"memory_stats": {
    "usage": 1814528,
    "max_usage": 6299648,
    "stats": {
        "active_anon": 606208,
        "active_file": 0,
        "cache": 0,
        "dirty": 0,
        "hierarchical_memory_limit": 134217728,
        "hierarchical_mems_w_limit": 268435456,
        "inactive_anon": 0,
        "inactive_file": 0,
        "mapped_file": 0,
        "pgfault": 5377,
        "pgmajfault": 0,
        "ppgpin": 3613,
        "ppgout": 3465,
        "rss": 606208,
        "rss_huge": 0,
        "total_active_anon": 606208,
        "total_active_file": 0,
        "total_cache": 0,
        "total_dirty": 0,
        "total_inactive_anon": 0,
        "total_inactive_file": 0,
        "total_mapped_file": 0,
        "total_pgfault": 5377,
        "total_pgmajfault": 0,
        "total_ppgpin": 3613,
        "total_ppgout": 3465,
        "total_rss": 606208,
        "total_rss_huge": 0,
    }
}
```

```
        "total_unevictable": 0,
        "total_writeback": 0,
        "unevictable": 0,
        "writeback": 0
    },
    "limit": 134217728
},
"name": "/ecs-curltest-26-curl-c2e5f6e0cf91b0bead01",
"id": "5fc21e5b015f899d22618f8aede80b6d70d71b2a75465ea49d9462c8f3d2d3af",
"networks": {
    "eth0": {
        "rx_bytes": 84,
        "rx_packets": 2,
        "rx_errors": 0,
        "rx_dropped": 0,
        "tx_bytes": 84,
        "tx_packets": 2,
        "tx_errors": 0,
        "tx_dropped": 0
    }
},
"network_rate_stats": {
    "rx_bytes_per_sec": 0,
    "tx_bytes_per_sec": 0
}
}
```

Task Metadata Endpoint version 3

Important

If you are using Amazon ECS tasks hosted on AWS Fargate, see [Task metadata endpoint version 3](#) in the *Amazon Elastic Container Service User Guide for AWS Fargate*.

Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called `ECS_CONTAINER_METADATA_URI` into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and [Docker stats](#) are available to tasks. For tasks that use the bridge network mode, network metrics are available when querying the `/stats` endpoints.

Enabling Task Metadata

The task metadata endpoint version 3 feature is enabled by default for tasks that use the Fargate launch type on platform version v1.3.0 or later and tasks that use the EC2 launch type and are launched on Amazon EC2 Linux infrastructure running at least version 1.21.0 of the Amazon ECS container agent or on Amazon EC2 Windows infrastructure running at least version 1.54.0 of the Amazon ECS container agent and use `awsvpc` network mode. For more information, see [Amazon ECS container agent versions \(p. 440\)](#).

You can add support for this feature on older container instances by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent \(p. 448\)](#).

Important

For tasks using the Fargate launch type and platform versions prior to v1.3.0, the task metadata version 2 endpoint is supported. For more information, see [Task Metadata Endpoint version 2 \(p. 498\)](#).

Task Metadata Endpoint version 3 Paths

The following task metadata endpoints are available to containers:

`#{ECS_CONTAINER_METADATA_URI}`

This path returns metadata JSON for the container.

`#{ECS_CONTAINER_METADATA_URI}/task`

This path returns metadata JSON for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see [Task Metadata JSON Response \(p. 494\)](#).

`#{ECS_CONTAINER_METADATA_URI}/taskWithTags`

This path returns the metadata for the task included in the /task endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API.

`#{ECS_CONTAINER_METADATA_URI}/stats`

This path returns Docker stats JSON for the specific Docker container. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

`#{ECS_CONTAINER_METADATA_URI}/task/stats`

This path returns Docker stats JSON for all of the containers associated with the task. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

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.

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.

Example Container Metadata Response

When querying the `#{ECS_CONTAINER_METADATA_URI}` endpoint you are returned only metadata about the container itself. The following is an example output.

```
{  
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",  
    "Name": "nginx-curl",  
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",  
    "Image": "nrqlngr/nginx-curl",  
    "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dc458dc0de47bc165",  
    "Labels": {  
        "com.amazonaws.ecs.cluster": "default",  
        "com.amazonaws.ecs.container-name": "nginx-curl",  
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",  
        "com.amazonaws.ecs.task-definition-family": "nginx",  
        "com.amazonaws.ecs.task-definition-version": "5"  
    }  
}
```

```

},
"DesiredStatus": "RUNNING",
"KnownStatus": "RUNNING",
"Limits": {
    "CPU": 512,
    "Memory": 512
},
"CreatedAt": "2018-02-01T20:55:10.554941919Z",
"StartedAt": "2018-02-01T20:55:11.064236631Z",
>Type": "NORMAL",
"Networks": [
    {
        "NetworkMode": "awsvpc",
        "IPv4Addresses": [
            "10.0.2.106"
        ]
    }
]
}

```

Example Task Metadata Response

When querying the `#{ECS_CONTAINER_METADATA_URI}/task` endpoint you are returned metadata about the task the container is part of. The following is an example output.

The following JSON response is for a single-container task.

```
{
    "Cluster": "default",
    "TaskARN": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
    "Family": "nginx",
    "Revision": "5",
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Containers": [
        {
            "DockerId": "731a0d6a3b4210e2448339bc7015aaa79bfe4fa256384f4102db86ef94cbbc4c",
            "Name": "~internal-ecs-pause",
            "DockerName": "ecs-nginx-5-internalecspause-acc699c0cbf2d6d11700",
            "Image": "amazon/amazon-ecs-pause:0.1.0",
            "ImageID": "",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "~internal-ecs-pause",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
                "com.amazonaws.ecs.task-definition-family": "nginx",
                "com.amazonaws.ecs.task-definition-version": "5"
            },
            "DesiredStatus": "RESOURCES_PROVISIONED",
            "KnownStatus": "RESOURCES_PROVISIONED",
            "Limits": {
                "CPU": 0,
                "Memory": 0
            },
            "CreatedAt": "2018-02-01T20:55:08.366329616Z",
            "StartedAt": "2018-02-01T20:55:09.058354915Z",
            "Type": "CNI_PAUSE",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.106"
                    ]
                }
            ]
        }
    ]
}
```

```

        ]
    }
],
{
    "DockerId": "43481a6ce4842eec8fe72fc28500c6b52edcc0917f105b83379f88cac1ff3946",
    "Name": "nginx-curl",
    "DockerName": "ecs-nginx-5-nginx-curl-ccccb9f49db0dfe0d901",
    "Image": "nrqlngr/nginx-curl",
    "ImageID": "sha256:2e00ae64383cfc865ba0a2ba37f61b50a120d2d9378559dc458dc0de47bc165",
    "Labels": {
        "com.amazonaws.ecs.cluster": "default",
        "com.amazonaws.ecs.container-name": "nginx-curl",
        "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
        "com.amazonaws.ecs.task-definition-family": "nginx",
        "com.amazonaws.ecs.task-definition-version": "5"
    },
    "DesiredStatus": "RUNNING",
    "KnownStatus": "RUNNING",
    "Limits": {
        "CPU": 512,
        "Memory": 512
    },
    "CreatedAt": "2018-02-01T20:55:10.554941919Z",
    "StartedAt": "2018-02-01T20:55:11.064236631Z",
    "Type": "NORMAL",
    "Networks": [
        {
            "NetworkMode": "awsvpc",
            "IPv4Addresses": [
                "10.0.2.106"
            ]
        }
    ],
    "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
    "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
    "AvailabilityZone": "us-east-2b"
}

```

Task Metadata Endpoint version 2

Beginning with version 1.17.0 of the Amazon ECS container agent, various task metadata and [Docker stats](#) are available to tasks that use the awsvpc network mode at an HTTP endpoint that is provided by the Amazon ECS container agent.

All containers belonging to tasks that are launched with the awsvpc network mode receive a local IPv4 address within a predefined link-local address range. When a container queries the metadata endpoint, the Amazon ECS container agent can determine which task the container belongs to based on its unique IP address, and metadata and stats for that task are returned.

Enabling task metadata

The task metadata version 2 feature is enabled by default for the following:

- Tasks using the Fargate launch type that use platform version v1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 169\)](#).
- Tasks using the EC2 launch type that also use the awsvpc network mode and are launched on Amazon EC2 Linux infrastructure running at least version 1.17.0 of the Amazon ECS container agent or on

Amazon EC2 Windows infrastructure running at least version 1.54.0 of the Amazon ECS container agent. For more information, see [Amazon ECS container agent versions \(p. 440\)](#).

You can add support for this feature on older container instances by updating the agent to the latest version. For more information, see [Updating the Amazon ECS container agent \(p. 448\)](#).

Task metadata endpoint paths

The following API endpoints are available to containers:

`169.254.170.2/v2/metadata`

This endpoint returns metadata JSON for the task, including a list of the container IDs and names for all of the containers associated with the task. For more information about the response for this endpoint, see [Task Metadata JSON Response \(p. 499\)](#).

`169.254.170.2/v2/metadata/<container-id>`

This endpoint returns metadata JSON for the specified Docker container ID.

`169.254.170.2/v2/metadata/taskWithTags`

This path returns the metadata for the task included in the `/task` endpoint in addition to the task and container instance tags that can be retrieved using the `ListTagsForResource` API.

`169.254.170.2/v2/stats`

This endpoint returns Docker stats JSON for all of the containers associated with the task. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

`169.254.170.2/v2/stats/<container-id>`

This endpoint returns Docker stats JSON for the specified Docker container ID. For more information about each of the returned stats, see [ContainerStats](#) in the Docker API documentation.

Task Metadata JSON Response

The following information is returned from the task metadata endpoint (`169.254.170.2/v2/metadata`) JSON response.

Cluster

The Amazon Resource Name (ARN) or short name of the Amazon ECS cluster to which the task belongs.

TaskARN

The 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.

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": "sha256:2e00ae64383fc865ba0a2ba37f61b50a120d2d9378559dc458dc0de47bc165",
            "Labels": {
                "com.amazonaws.ecs.cluster": "default",
                "com.amazonaws.ecs.container-name": "nginx-curl",
                "com.amazonaws.ecs.task-arn": "arn:aws:ecs:us-
east-2:012345678910:task/9781c248-0edd-4cdb-9a93-f63cb662a5d3",
                "com.amazonaws.ecs.task-definition-family": "nginx",
                "com.amazonaws.ecs.task-definition-version": "5"
            },
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Limits": {
                "CPU": 512,
                "Memory": 512
            },
            "CreatedAt": "2018-02-01T20:55:10.554941919Z",
            "StartedAt": "2018-02-01T20:55:11.064236631Z",
            "Type": "NORMAL",
            "Networks": [
                {
                    "NetworkMode": "awsvpc",
                    "IPv4Addresses": [
                        "10.0.2.106"
                    ]
                }
            ]
        }
    ]
}

```

```
        ],
        "PullStartedAt": "2018-02-01T20:55:09.372495529Z",
        "PullStoppedAt": "2018-02-01T20:55:10.552018345Z",
        "AvailabilityZone": "us-east-2b"
    }
```

Amazon ECS container agent introspection

The Amazon ECS container agent provides an API operation for gathering details about the container instance on which the agent is running and the associated tasks running on that instance. You can use the `curl` command from within the container instance to query the Amazon ECS container agent (port 51678) and return container instance metadata or task information.

Important

Your container instance must have an IAM role that allows access to Amazon ECS in order to retrieve the metadata. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

To view container instance metadata, log in to your container instance via SSH and run the following command. Metadata includes the container instance ID, the Amazon ECS cluster in which the container instance is registered, and the Amazon ECS container agent version information.

```
curl -s http://localhost:51678/v1/metadata | python -mjson.tool
```

Output:

```
{
    "Cluster": "cluster_name",
    "ContainerInstanceArn": "arn:aws:ecs:region:aws_account_id:container-
instance/cluster_name/container_instance_id",
    "Version": "Amazon ECS Agent - v1.30.0 (02ff320c)"
}
```

To view information about all of the tasks that are running on a container instance, log in to your container instance via SSH and run the following command:

```
curl http://localhost:51678/v1/tasks
```

Output:

```
{
    "Tasks": [
        {
            "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/example5-58ff-46c9-
ae05-543f8example",
            "DesiredStatus": "RUNNING",
            "KnownStatus": "RUNNING",
            "Family": "hello_world",
            "Version": "8",
            "Containers": [
                {
                    "DockerId": "9581a69a761a557fbfce1d0f6745e4af5b9dbfb86b6b2c5c4df156f1a5932ff1",
                    "DockerName": "ecs-hello_world-8-mysql-fcae8ac8f9f1d89d8301",
                    "Name": "mysql"
                }
            ]
        }
    ]
}
```

```
        },
        {
            "DockerId": "bf25c5c5b2d4dba68846c7236e75b6915e1e778d31611e3c6a06831e39814a15",
            "DockerName": "ecs-hello_world-8-wordpress-e8bfdd9b488dff36c00",
            "Name": "wordpress"
        }
    ]
}
```

You can view information for a particular task that is running on a container instance. To specify a specific task or container, append one of the following to the request:

- The task ARN (`?taskarn=task_arn`)
- The Docker ID for a container (`?dockerid=docker_id`)

To get task information with a container's Docker ID, log in to your container instance via SSH and run the following command.

Note

Amazon ECS container agents before version 1.14.2 require full Docker container IDs for the introspection API, not the short version that is shown with `docker ps`. You can get the full Docker ID for a container by running the `docker ps --no-trunc` command on the container instance.

```
curl http://localhost:51678/v1/tasks?dockerid=79c796ed2a7f
```

Output:

```
{
    "Arn": "arn:aws:ecs:us-west-2:012345678910:task/default/e01d58a8-151b-40e8-
bc01-22647b9ecfec",
    "Containers": [
        {
            "DockerId": "79c796ed2a7f864f485c76f83f3165488097279d296a7c05bd5201a1c69b2920",
            "DockerName": "ecs-nginx-efs-2-nginx-9ac0808dd0afa495f001",
            "Name": "nginx"
        }
    ],
    "DesiredStatus": "RUNNING",
    "Family": "nginx-efs",
    "KnownStatus": "RUNNING",
    "Version": "2"
}
```

HTTP proxy configuration

You can configure your Amazon ECS container instances to use an HTTP proxy for both the Amazon ECS container agent and the Docker daemon. This is useful if your container instances do not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance. The process differs for Linux and Windows instances, so be sure to read the appropriate section below for your application.

Topics

- [Amazon Linux container instance configuration \(p. 505\)](#)

- [Windows container instance configuration \(p. 507\)](#)

Amazon Linux container instance configuration

To configure your Amazon ECS Linux container instance to use an HTTP proxy, set the following variables in the relevant files at launch time (with Amazon EC2 user data). You could also manually edit the configuration file and restart the agent afterwards.

/etc/ecs/ecs.config (Amazon Linux 2 and Amazon Linux AMI)

```
HTTP_PROXY=10.0.0.131:3128
```

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the ECS agent to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

```
NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock
```

Set this value to 169.254.169.254,169.254.170.2,/var/run/docker.sock to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

/etc/systemd/system/ecs.service.d/http-proxy.conf (Amazon Linux 2 only)

```
Environment="HTTP_PROXY=10.0.0.131:3128/"
```

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for ecs-init to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

```
Environment="NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock"
```

Set this value to 169.254.169.254,169.254.170.2,/var/run/docker.sock to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

/etc/init/ecs.override (Amazon Linux AMI only)

```
env HTTP_PROXY=10.0.0.131:3128
```

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for ecs-init to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

```
env NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock
```

Set this value to 169.254.169.254,169.254.170.2,/var/run/docker.sock to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

/etc/systemd/system/docker.service.d/http-proxy.conf (Amazon Linux 2 only)

```
Environment="HTTP_PROXY=http://10.0.0.131:3128"
```

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Docker daemon to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

```
Environment="NO_PROXY=169.254.169.254"
```

Set this value to 169.254.169.254 to filter EC2 instance metadata from the proxy.

/etc/sysconfig/docker (Amazon Linux AMI and Amazon Linux 2 only)

```
export HTTP_PROXY=http://10.0.0.131:3128
```

Set this value to the hostname (or IP address) and port number of an HTTP proxy to use for the Docker daemon to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

```
export NO_PROXY=169.254.169.254,169.254.170.2
```

Set this value to 169.254.169.254 to filter EC2 instance metadata from the proxy.

Setting these environment variables in the above files only affects the Amazon ECS container agent, `ecs-init`, and the Docker daemon. They do not configure any other services (such as `yum`) to use the proxy.

Example Amazon Linux HTTP proxy user data script

The example user data `cloud-boothook` script below configures the Amazon ECS container agent, `ecs-init`, the Docker daemon, and `yum` to use an HTTP proxy that you specify. You can also specify a cluster into which the container instance registers itself.

To use this script when you launch a container instance, follow the steps in [Launching an Amazon ECS Linux container instance \(p. 364\)](#), and in [Step 6.g \(p. 365\)](#). Then, copy and paste the `cloud-boothook` script below into the **User data** field (be sure to substitute the red example values with your own proxy and cluster information).

Note

The user data script below only supports Amazon Linux 2 and Amazon Linux AMI variants of the Amazon ECS-optimized AMI.

```
#cloud-boothook
# Configure Yum, the Docker daemon, and the ECS agent to use an HTTP proxy

# Specify proxy host, port number, and ECS cluster name to use
PROXY_HOST=10.0.0.131
PROXY_PORT=3128
CLUSTER_NAME=proxy-test

if grep -q 'Amazon Linux release 2' /etc/system-release ; then
    OS=AL2
    echo "Setting OS to Amazon Linux 2"
elif grep -q 'Amazon Linux AMI' /etc/system-release ; then
    OS=ALAMI
    echo "Setting OS to Amazon Linux AMI"
else
    echo "This user data script only supports Amazon Linux 2 and Amazon Linux AMI."
fi

# Set Yum HTTP proxy
if [ ! -f /var/lib/cloud/instance/se�/config_yum_http_proxy ]; then
    echo "proxy=http://$PROXY_HOST:$PROXY_PORT" >> /etc/yum.conf
    echo "$$: $(date +%-s.%N | cut -b1-13)" > /var/lib/cloud/instance/se�/
config_yum_http_proxy
fi

# Set Docker HTTP proxy (different methods for Amazon Linux 2 and Amazon Linux AMI)
# Amazon Linux 2
if [ $OS == "AL2" ] && [ ! -f /var/lib/cloud/instance/se�/config_docker_http_proxy ]; then
    mkdir /etc/systemd/system/docker.service.d
    cat <<EOF > /etc/systemd/system/docker.service.d/http-proxy.conf
[Service]
Environment="HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/"
Environment="HTTPS_PROXY=https://$PROXY_HOST:$PROXY_PORT/"
Environment="NO_PROXY=169.254.169.254,169.254.170.2"
EOF
    systemctl daemon-reload
    if [ "$(systemctl is-active docker)" == "active" ]
    then
        systemctl restart docker
    fi
fi
```

```

        echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_docker_http_proxy
fi
# Amazon Linux AMI
if [ $OS == "ALAMI" ] && [ ! -f /var/lib/cloud/instance/sem/config_docker_http_proxy ];
then
    echo "export HTTP_PROXY=http://$PROXY_HOST:$PROXY_PORT/" >> /etc/sysconfig/docker
    echo "export HTTPS_PROXY=https://$PROXY_HOST:$PROXY_PORT/" >> /etc/sysconfig/docker
    echo "export NO_PROXY=169.254.169.254,169.254.170.2" >> /etc/sysconfig/docker
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/
config_docker_http_proxy
fi

# Set ECS agent HTTP proxy
if [ ! -f /var/lib/cloud/instance/sem/config_ecs-agent_http_proxy ]; then
    cat <<EOF > /etc/ecs/ecs.config
ECS_CLUSTER=$CLUSTER_NAME
HTTP_PROXY=$PROXY_HOST:$PROXY_PORT
NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock
EOF
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_ecs-
agent_http_proxy
fi

# Set ecs-init HTTP proxy (different methods for Amazon Linux 2 and Amazon Linux AMI)
# Amazon Linux 2
if [ $OS == "AL2" ] && [ ! -f /var/lib/cloud/instance/sem/config_ecs-init_http_proxy ];
then
    mkdir /etc/systemd/system/ecs.service.d
    cat <<EOF > /etc/systemd/system/ecs.service.d/http-proxy.conf
[Service]
Environment="HTTP_PROXY=$PROXY_HOST:$PROXY_PORT/"
Environment="NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock"
EOF
    systemctl daemon-reload
    if [ "$(systemctl is-active ecs)" == "active" ]; then
        systemctl restart ecs
    fi
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_ecs-
init_http_proxy
fi
# Amazon Linux AMI
if [ $OS == "ALAMI" ] && [ ! -f /var/lib/cloud/instance/sem/config_ecs-init_http_proxy ];
then
    cat <<EOF > /etc/init/ecs.override
env HTTP_PROXY=$PROXY_HOST:$PROXY_PORT
env NO_PROXY=169.254.169.254,169.254.170.2,/var/run/docker.sock
EOF
    echo "$$: $(date +%s.%N | cut -b1-13)" > /var/lib/cloud/instance/sem/config_ecs-
init_http_proxy
fi

```

Windows container instance configuration

To configure your Amazon ECS Windows container instance to use an HTTP proxy, set the following variables at launch time (with Amazon EC2 user data).

```
[Environment]::SetEnvironmentVariable("HTTP_PROXY",
"http://proxy.mydomain:port", "Machine")
```

Set `HTTP_PROXY` to the hostname (or IP address) and port number of an HTTP proxy to use for the ECS agent to connect to the internet. For example, your container instances may not have external network access through an Amazon VPC internet gateway, NAT gateway, or instance.

```
[Environment]::SetEnvironmentVariable("NO_PROXY",
"169.254.169.254,169.254.170.2,\.\.\pipe\docker_engine", "Machine")
```

Set NO_PROXY to 169.254.169.254,169.254.170.2,\.\.\pipe\docker_engine to filter EC2 instance metadata, IAM roles for tasks, and Docker daemon traffic from the proxy.

Example Windows HTTP proxy user data script

The example user data PowerShell script below configures the Amazon ECS container agent and the Docker daemon to use an HTTP proxy that you specify. You can also specify a cluster into which the container instance registers itself.

To use this script when you launch a container instance, follow the steps in [Launching an Amazon ECS Linux container instance \(p. 364\)](#). Just copy and paste the PowerShell script below into the **User data** field (be sure to substitute the red example values with your own proxy and cluster information).

Note

The `-EnableTaskIAMRole` option is required to enable IAM roles for tasks. For more information, see [Windows IAM roles for tasks \(p. 846\)](#).

```
<powershell>
Import-Module ECSTools

$proxy = "http://proxy.mydomain:port"
[Environment]::SetEnvironmentVariable("HTTP_PROXY", $proxy, "Machine")
[Environment]::SetEnvironmentVariable("NO_PROXY", "169.254.169.254,169.254.170.2,\.\.\pipe\docker_engine", "Machine")

Restart-Service Docker
Initialize-ECSAgent -Cluster MyCluster -EnableTaskIAMRole
</powershell>
```

Scheduling Amazon ECS tasks

Amazon Elastic Container Service (Amazon ECS) is a shared state, optimistic concurrency system that provides flexible scheduling capabilities for your tasks and containers. The Amazon ECS schedulers 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 \(p. 533\)](#).
- **DAEMON**—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see [Daemon \(p. 532\)](#).

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 \(p. 531\)](#).

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 \(p. 510\)](#) 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 a CloudWatch Events rule. You can run tasks for a backup operation or a log scan. The CloudWatch Events rule that you create can run one or more tasks in your cluster at specified times. Your scheduled event rule 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 \(p. 524\)](#).

Custom schedulers

With Amazon ECS, you can create your own schedulers or use third-party schedulers. [Blox](#) is an open-source project that gives you more control over how your containerized applications run on Amazon ECS. You can use it to build schedulers and integrate third-party schedulers with Amazon ECS. At the same time, you can also use Amazon ECS to manage and scale your clusters. Custom schedulers use the [StartTask](#) API operation to place tasks on specific container instances within your cluster.

Note

Custom schedulers are only compatible with tasks hosted on EC2 instances. If you use Amazon ECS on Fargate, the StartTask API doesn't work.

Task placement

Use [RunTask](#) and [CreateService](#) actions to specify task placement constraints and task placement strategies. These customize how Amazon ECS places and runs your tasks. For more information, see [Amazon ECS task placement \(p. 513\)](#).

Contents

- [Run a standalone task \(p. 510\)](#)
- [Amazon ECS task placement \(p. 513\)](#)
- [Scheduled tasks \(p. 524\)](#)
- [Task lifecycle \(p. 527\)](#)
- [Creating a scheduled task using the AWS CLI \(p. 529\)](#)

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 \(p. 531\)](#).

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.

New console

To run a standalone task using the new console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster to run the standalone task in.
3. From the **Tasks** tab, choose **Run new task**.
4. 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. 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, that capacity provider strategy is 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 that uses the **FARGATE** capacity provider is 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. For this, you must manually define the strategy.
 - If your cluster doesn't have a default capacity provider strategy defined and no capacity providers are added to the cluster, the Fargate launch type is selected.
5. For **Application type**, choose **Task**.
 6. For **Task definition**, choose the task definition family and revision to use.

Important
The console validates the selection to ensure that the selected task definition family and revision is compatible with the defined compute configuration.
 7. For **Desired tasks**, specify the number of tasks to run in the cluster.
 8. The **Networking** section can be expanded to define the Amazon VPC, subnet, and security group configuration if your task requires it. Task definitions that use the `awsvpc` network mode 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. If your application requires it, create a new security group.
 9. (Optional) The **Tags** section can be expanded to add tags, as key-value pairs, to the service.
 10. Choose **Deploy**.

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 \(p. 178\)](#).
 - 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 \(p. 247\)](#).
 - b. For **Cluster**, choose the cluster to use.
 - c. For **Number of tasks**, enter the number of tasks to launch with this task definition.
 - d. 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) For **Task Placement**, you can specify how tasks are placed using task placement strategies and constraints. Choose from the following options:

- **AZ Balanced Spread** - Distribute tasks across Availability Zones and across container instances in the Availability Zone.
- **AZ Balanced BinPack** - Distribute tasks across Availability Zones and across container instances with the least available memory.
- **BinPack** - Distribute tasks based on the least available amount of CPU or memory.
- **One Task Per Host** - Place, at most, one task from the service on each container instance.
- **Custom** - Define your own task placement strategy. For examples, see [Amazon ECS task placement \(p. 513\)](#).

For more information, see [Amazon ECS task placement \(p. 513\)](#).

7. (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 \(p. 699\)](#).

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 \(p. 702\)](#).

- 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 \(p. 691\)](#).

- 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`), the format is an unquoted string. This string is surrounded with double quotation marks (" ") and passed 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 (" ")).



This environment variable override is sent to the container in the following format:

```
MY_ENV_VAR="This variable contains a string."
```

8. In the **Task tagging configuration** section, complete the following steps:
 - 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](#).
 - 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.

9. 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](#).
10. 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 \(p. 815\)](#) in the troubleshooting section.

Amazon ECS task placement

When a task that uses the EC2 launch type is launched, Amazon ECS must determine where to place the task based on the requirements specified in the task definition, such as CPU and memory. Similarly, when you scale down the task count, Amazon ECS must determine which tasks to terminate. You can apply task placement strategies and constraints to customize how Amazon ECS places and terminates tasks. Task placement strategies and constraints aren't supported for tasks using the Fargate launch type. By default, Fargate tasks are spread across Availability Zones. With all other tasks, default task placement

strategies depend on whether you're running tasks manually or within a service. For more information, see [Scheduling Amazon ECS tasks \(p. 509\)](#).

A *task placement strategy* is an algorithm for selecting instances for task placement or tasks for termination. For example, Amazon ECS can select instances at random, or it can select instances such that tasks are distributed evenly across a group of instances.

A *task placement constraint* is a rule that's considered during task placement. For example, you can use constraints to place tasks based on Availability Zone or instance type. You can also associate *attributes*, which are name/value pairs, with your container instances and then use a constraint to place tasks based on attribute.

Note

Task placement strategies are a best effort. Amazon ECS still attempts to place tasks even when the most optimal placement option is unavailable. However, task placement constraints are binding, and they can prevent task placement.

You can use task placement strategies and constraints together. For example, you can use a task placement strategy and a task placement constraint to distribute tasks across Availability Zones and bin pack tasks based on memory within each Availability Zone, but only for G2 instances.

When Amazon ECS places tasks, it uses the following process to select container instances:

1. Identify the instances that satisfy the CPU, memory, and port requirements in the task definition.
2. Identify the instances that satisfy the task placement constraints.
3. Identify the instances that satisfy the task placement strategies.
4. Select the instances for task placement.

Contents

- [Task groups \(p. 514\)](#)
- [Amazon ECS task placement strategies \(p. 515\)](#)
- [Amazon ECS task placement constraints \(p. 516\)](#)
- [Cluster query language \(p. 520\)](#)

Task groups

You can identify a set of related tasks as a *task group*. All tasks with the same task group name are considered as a set when using the *spread* task placement strategy. For example, suppose that you're running different applications in one cluster, such as databases and web servers. To ensure that your databases are balanced across Availability Zones, add them to a task group named `databases` and then use the *spread* task placement strategy. For more information, see [Amazon ECS task placement strategies \(p. 515\)](#).

Task groups can also be used as a task placement constraint. For specifically, they can be used in specifying the `memberOf` constraint to ensure that tasks in the same task group are placed on the same instances. For an example, see [Example constraints \(p. 520\)](#).

By default, standalone tasks use the task definition family name (for example, `family:my-task-definition`) as the task group name if a custom task group name isn't specified. Tasks launched as part of a service use the service name as the task group name and can't be changed.

The following requirements for the task group name must be considered.

- A task group name must be 255 or fewer characters.

- Each task can be in exactly one group.
- After launching a task, you can't modify its task group.

Amazon ECS task placement strategies

A *task placement strategy* is an algorithm for selecting instances for task placement or tasks for termination. Task placement strategies can be specified when either running a task or creating a new service. The task placement strategies can be updated for existing services as well. For more information, see [Amazon ECS task placement \(p. 513\)](#).

Strategy types

Amazon ECS supports the following task placement strategies:

binpack

Tasks are placed on container instances so as to leave the least amount of unused CPU or memory. This strategy minimizes the number of container instances in use.

When this strategy is used and a scale-in action is taken, Amazon ECS terminates tasks. It does this based on the amount of resources that are left on the container instance after the task is terminated. The container instance that has the most available resources left after task termination has that task terminated.

random

Tasks are placed randomly.

spread

Tasks are placed evenly based on the specified value. Accepted values are `instanceId` (or `host`, which has the same effect), or any platform or custom attribute that's applied to a container instance, such as `attribute:ecs.availability-zone`.

Service tasks are spread based on the tasks from that service. Standalone tasks are spread based on the tasks from the same task group. For more information about task groups, see [Task groups \(p. 514\)](#).

When the `spread` strategy is used and a scale-in action is taken, Amazon ECS selects tasks to terminate that maintain a balance across Availability Zones. Within an Availability Zone, tasks are selected at random.

Example strategies

You can specify task placement strategies with the following actions: [CreateService](#), [UpdateService](#), and [RunTask](#).

The following strategy distributes tasks evenly across Availability Zones.

```
"placementStrategy": [
  {
    "field": "attribute:ecs.availability-zone",
    "type": "spread"
  }
]
```

The following strategy distributes tasks evenly across all instances.

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

The following strategy bin packs tasks based on memory.

```
"placementStrategy": [
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

The following strategy places tasks randomly.

```
"placementStrategy": [
    {
        "type": "random"
    }
]
```

The following strategy distributes tasks evenly across Availability Zones and then distributes tasks evenly across the instances within each Availability Zone.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

The following strategy distributes tasks evenly across Availability Zones and then bin packs tasks based on memory within each Availability Zone.

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

Amazon ECS task placement constraints

A *task placement constraint* is a rule that's considered during task placement. Task placement constraints can be specified when either running a task or creating a new service. The task placement constraints can be updated for existing services as well. For more information, see [Amazon ECS task placement \(p. 513\)](#).

Constraint types

Amazon ECS supports the following types of task placement constraints:

`distinctInstance`

Place each task on a different container instance. This task placement constraint can be specified when either running a task or creating a new service.

`memberOf`

Place tasks on container instances that satisfy an expression. For more information about the expression syntax for constraints, see [Cluster query language \(p. 520\)](#).

The `memberOf` task placement constraint can be specified with the following actions:

- Running a task
- Creating a new service
- Creating a new task definition
- Creating a new revision of an existing task definition

Attributes

You can add custom metadata to your container instances, known as *attributes*. Each attribute has a name and an optional string value. You can use the built-in attributes provided by Amazon ECS or define custom attributes.

Built-in attributes

Amazon ECS automatically applies the following attributes to your container instances.

`ecs.ami-id`

The ID of the AMI used to launch the instance. An example value for this attribute is `ami-1234abcd`.

`ecs.availability-zone`

The Availability Zone for the instance. An example value for this attribute is `us-east-1a`.

`ecs.instance-type`

The instance type for the instance. An example value for this attribute is `g2.2xlarge`.

`ecs.os-type`

The operating system for the instance. The possible values for this attribute are `linux` and `windows`.

`ecs.cpu-architecture`

The CPU architecture for the instance. Example values for this attribute are `x86_64` and `arm64`.

`ecs.vpc-id`

The VPC the instance was launched into. An example value for this attribute is `vpc-1234abcd`.

`ecs.subnet-id`

The subnet the instance is using. An example value for this attribute is `subnet-1234abcd`.

Optional attributes

Amazon ECS may add the following attributes to your container instances.

`ecs.awsVpcTrunkId`

If this attribute exists, the instance has a trunk network interface. For more information, see [Elastic network interface trunking \(p. 372\)](#).

`ecs.outpostArn`

If this attribute exists, it contains the Amazon Resource Name (ARN) of the Outpost. For more information, see [the section called "Amazon Elastic Container Service on AWS Outposts" \(p. 722\)](#).

`ecs.capability.external`

If this attribute exists, the instance is identified as an external instance. For more information, see [External instances \(Amazon ECS Anywhere\) \(p. 414\)](#).

Custom attributes

You can apply custom attributes to your container instances. For example, you can define an attribute with the name "stack" and a value of "prod".

When specifying custom attributes, you must consider the following.

- The name must contain between 1 and 128 characters and may contain letters (uppercase and lowercase), numbers, hyphens, underscores, forward slashes, back slashes, or periods.
- The value must contain between 1 and 128 characters and may contain letters (uppercase and lowercase), numbers, hyphens, underscores, periods, at signs (@), forward slashes, back slashes, colons, or spaces. The value can't contain any leading or trailing whitespace.

Adding an attribute

You can add custom attributes at instance registration time using the container agent or manually, using the AWS Management Console. For more information about using the container agent, see [Amazon ECS container agent configuration parameters \(p. 462\)](#).

To add custom attributes using the console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Clusters** and select a cluster.
3. On the **ECS Instances** tab, select the check box for the container instance.
4. Choose **Actions, View/Edit Attributes**.
5. For each attribute, do the following:
 - a. Choose **Add attribute**.
 - b. Enter a name and a value for the attribute and choose the checkmark icon.
6. When you're finished adding attributes, choose **Close**.

Adding custom attributes using the AWS CLI

The following examples demonstrate how to add custom attributes using the `put-attributes` command.

Example: Single Attribute

The following example adds the custom attribute "stack=prod" to the specified container instance in the default cluster.

```
aws ecs put-attributes --attributes name=stack,value=prod,targetId=arn
```

Example: Multiple Attributes

The following example adds the custom attributes "stack=prod" and "project=a" to the specified container instance in the default cluster.

```
aws ecs put-attributes --attributes name=stack,value=prod,targetId=arn
name=project,value=a,targetId=arn
```

Filtering by attribute

You can apply a filter for your container instances, allowing you to see custom attributes.

Filter container instances by attribute using the console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. Choose a cluster that has container instances.
3. Choose **ECS Instances**.
4. Set column visibility preferences by choosing the gear icon (⚙) and selecting the attributes to display. This setting persists across all container clusters associated with your account.
5. Using the **Filter by attributes** text field, enter or select the attributes you want to filter by. The format must be *AttributeName:AttributeValue*.

For **Filter by attributes**, enter or select the attributes to filter by. After you select the attribute name, you're prompted for the attribute value.
6. Add additional attributes to the filter as needed. Remove an attribute by choosing the X next to it.

Filter container instances by attribute using the AWS CLI

The following examples demonstrate how to filter container instances by attribute using the `list-container-instances` command. For more information about the filter syntax, see [Cluster query language \(p. 520\)](#).

Example: Built-in attribute

The following example uses built-in attributes to list the g2.2xlarge instances.

```
aws ecs list-container-instances --filter "attribute:ecs.instance-type == g2.2xlarge"
```

Example: Custom attribute

The following example lists the instances with the custom attribute "stack=prod".

```
aws ecs list-container-instances --filter "attribute:stack == prod"
```

Example: Exclude an attribute value

The following example lists the instances with the custom attribute "stack" unless the attribute value is "prod".

```
aws ecs list-container-instances --filter "attribute:stack != prod"
```

Example: Multiple attribute values

The following example uses built-in attributes to list the instances of type `t2.small` or `t2.medium`.

```
aws ecs list-container-instances --filter "attribute:ecs.instance-type in [t2.small, t2.medium]"
```

Example: Multiple attributes

The following example uses built-in attributes to list the T2 instances in the us-east-1a Availability Zone.

```
aws ecs list-container-instances --filter "attribute:ecs.instance-type =~ t2.* and attribute:ecs.availability-zone == us-east-1a"
```

Example constraints

The following are task placement constraint examples.

This example uses the `memberOf` constraint to place tasks on T2 instances. It can be specified with the following actions: [CreateService](#), [UpdateService](#), [RegisterTaskDefinition](#), and [RunTask](#).

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-type =~ t2.*",
        "type": "memberOf"
    }
]
```

The example uses the `memberOf` constraint to place tasks on instances with other tasks in the databases task group, respecting any task placement strategies that are also specified. For more information about task groups, see [Task groups \(p. 514\)](#). It can be specified with the following actions: [CreateService](#), [UpdateService](#), [RegisterTaskDefinition](#), and [RunTask](#).

```
"placementConstraints": [
    {
        "expression": "task:group == databases",
        "type": "memberOf"
    }
]
```

The `distinctInstance` constraint places each task in the group on a different instance. It can be specified with the following actions: [CreateService](#), [UpdateService](#), and [RunTask](#)

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

Cluster query language

Cluster queries are expressions that enable you to group objects. For example, you can group container instances by attributes such as Availability Zone, instance type, or custom metadata. For more information, see [Attributes \(p. 517\)](#).

After you have defined a group of container instances, you can customize Amazon ECS to place tasks on container instances based on group. For more information, see [Run a standalone task \(p. 510\)](#) and [Creating an Amazon ECS service \(p. 546\)](#). You can also apply a group filter when listing container instances. For more information, see [Filtering by attribute \(p. 519\)](#).

Expression syntax

Expressions have the following syntax:

```
subject operator [argument]
```

Subject

The attribute or field to be evaluated.

`agentConnected`

Select container instances by their Amazon ECS container agent connection status. You can use this filter to search for instances with container agents that are disconnected.

Valid operators: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`agentVersion`

Select container instances by their Amazon ECS container agent version. You can use this filter to find instances that are running outdated versions of the Amazon ECS container agent.

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`attribute:attribute-name`

Select container instances by attribute. For more information, see [Attributes \(p. 517\)](#).

`ec2InstanceId`

Select container instances by their Amazon EC2 instance ID.

Valid operators: equals (==), not_equals (!=), in, not_in (!in), matches (=~), not_matches (!~)

`registeredAt`

Select container instances by their container instance registration date. You can use this filter to find newly registered instances or instances that are very old.

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

Valid date formats: 2018-06-18T22:28:28+00:00, 2018-06-18T22:28:28Z, 2018-06-18T22:28:28, 2018-06-18

`runningTasksCount`

Select container instances by number of running tasks. You can use this filter to find instances that are empty or near empty (few tasks running on them).

Valid operators: equals (==), not_equals (!=), greater_than (>), greater_than_equal (>=), less_than (<), less_than_equal (<=)

`task:group`

Select container instances by task group. For more information, see [Task groups \(p. 514\)](#).

Operator

The comparison operator. The following operators are supported.

Operator	Description
<code>==, equals</code>	String equality
<code>!=, not_equals</code>	String inequality
<code>>, greater_than</code>	Greater than
<code>>=, greater_than_equal</code>	Greater than or equal to
<code><, less_than</code>	Less than
<code><=, less_than_equal</code>	Less than or equal to
<code>exists</code>	Subject exists
<code>!exists, not_exists</code>	Subject doesn't exist
<code>in</code>	Value in argument list
<code>!in, not_in</code>	Value not in argument list
<code>=~, matches</code>	Pattern match
<code>!~, not_matches</code>	Pattern mismatch

Note

A single expression can't contain parentheses. However, parentheses can be used to specify precedence in compound expressions.

Argument

For many operators, the argument is a literal value.

The `in` and `not_in` operators expect an argument list as the argument. You specify an argument list as follows:

`[argument1, argument2, ..., argumentN]`

The `matches` and `not_matches` operators expect an argument that conforms to the Java regular expression syntax. For more information, see [java.util.regex.Pattern](#).

Compound expressions

You can combine expressions using the following Boolean operators:

- `&&`, and
- `||`, or
- `!`, not

You can specify precedence using parentheses:

`(expression1 or expression2) and expression3`

Example expressions

The following are example expressions.

Example: String Equality

The following expression selects instances with the specified instance type.

```
attribute:ecs.instance-type == t2.small
```

Example: Argument List

The following expression selects instances in the us-east-1a or us-east-1b Availability Zone.

```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

Example: Compound Expression

The following expression selects G2 instances that aren't in the us-east-1d Availability Zone.

```
attribute:ecs.instance-type =~ g2.* and attribute:ecs.availability-zone != us-east-1d
```

Example: Task Affinity

The following expression selects instances that are hosting tasks in the service:production group.

```
task:group == service:production
```

Example: Task Anti-Affinity

The following expression selects instances that aren't hosting tasks in the database group.

```
not(task:group == database)
```

Example: Running task count

The following expression selects instances that are only running one task.

```
runningTasksCount == 1
```

Example: Amazon ECS container agent version

The following expression selects instances that are running a container agent version below 1.14.5.

```
agentVersion < 1.14.5
```

Example: Instance registration time

The following expression selects instances that were registered before February 13, 2018.

```
registeredAt < 2018-02-13
```

Example: Amazon EC2 instance ID

The following expression selects instances with the following Amazon EC2 instance IDs.

```
ec2InstanceId in ['i-abcd1234', 'i-wxyx7890']
```

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*.

The EventBridge documentation includes a tutorial for creating a scheduled task based on a file being uploaded to an Amazon S3 bucket. For more information, see [Tutorial: Run an Amazon ECS task when a file is uploaded to an Amazon S3 bucket](#) in the *Amazon EventBridge User Guide*.

Contents

- [Create a scheduled task \(p. 524\)](#)
- [View your scheduled tasks \(p. 526\)](#)
- [Edit a scheduled task \(p. 526\)](#)

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**, **Create rule**.
3. 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.

4. For a scheduled task that runs on a schedule, do the following. To create a scheduled task that runs based on an event, skip to the next step.
 - a. For **Define pattern**, choose **Schedule**.
 - b. Either choose **Fixed rate of** and specify how often the task is to run, or choose **Cron expression** and specify a cron expression that defines when the task is to be triggered.
 - c. For **Select event bus**, choose **AWS default event bus**. You can only create scheduled rules on the default event bus.
5. For a scheduled task that runs based on an event, do the following. If you created your rule based on a schedule, skip to the next step.
 - a. For **Define pattern**, choose **Event pattern**.
 - b. Choose **Pre-defined pattern by service**.
 - c. For **Service provider**, choose **AWS**.
 - d. For **Service name**, choose the name of the service that emits the event.

- e. For **Event type**, choose **All Events** or choose the type of event to use for this rule. If you choose **All Events**, all events emitted by this AWS service match the rule.

To customize the event pattern, choose **Edit**, make your changes, and then choose **Save**.
 - f. For **Select 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. For **Select targets**, choose **ECS task**.
 7. For **Cluster**, select an Amazon ECS cluster.
 8. For **Task definition**, select a task definition family.
 9. For **Task definition revision**, select either **Latest** or **Revision** and select a specific task definition revision to use.
 10. For **Count**, specify the desired number of tasks to run.
 11. The **Compute options** section can be expanded to change the default compute options for the scheduled task.
 - a. Choose whether your scheduled task uses a **Capacity provider strategy** or **Launch type**.
 - b. To use a capacity provider strategy, choose **Use cluster default** to use the cluster's default capacity provider strategy. If your cluster doesn't have a default capacity provider strategy, or to use a custom strategy, choose **Use custom**, **Add capacity provider strategy** and define your custom capacity provider strategy by specifying a **Capacity provider**, **Base**, and **Weight**. In order for a capacity provider to be used in a strategy, it must be associated with the cluster. For more information about capacity provider strategies, see [Amazon ECS capacity providers \(p. 178\)](#).
 - c. To use a launch type instead, specify **Launch type**, choose the launch type to use.
 - d. (Optional) When the Fargate launch type is specified, for **Platform version**, specify the platform version to use. If a platform version isn't specified, the **LATEST** platform version is used by default.
 12. (Optional) Expand **Configure network configuration** to specify a network configuration. This is required for tasks hosted on Fargate and for tasks using the `awsvpc` network mode.
 - a. For **Subnets**, specify one or more subnet IDs.
 - b. For **Security groups**, specify one or more security group IDs.
 - c. For **Auto-assign public IP**, specify whether to assign a public IP address from your subnet to the task.
 13. (Optional) Expand **Configure additional properties** to specify the following additional parameters for your tasks.
 - a. 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.
 - b. For **Placement constraint**, choose **Add placement constraint**. Select the **Type** for the placement constraint and then enter an expression. For more information, see [Amazon ECS task placement constraints \(p. 516\)](#).

Note

Task placement constraints aren't supported for tasks hosted on Fargate.

- c. For **Placement strategy**, choose **Add placement strategy**. Select the **Type** for the placement strategy and then enter an expression. Repeat this process for each placement strategy to add. For more information, see [Amazon ECS task placement strategies \(p. 515\)](#).

Note

Task placement strategies aren't supported for tasks hosted on Fargate.

- d. For **Tags**, choose **Add tag** to associate key value pair tags for the task.

- e. For **Configure managed tags**, choose **Enable managed tags** to have Amazon ECS add tags that can be used when reviewing cost allocation in your Cost and Usage Report. For more information, see [Tagging your resources for billing \(p. 606\)](#).
- f. For **Configure execute command**, choose **Enable execute command** to enable the ECS Exec functionality for the task. For more information, see [Using Amazon ECS Exec for debugging \(p. 805\)](#).
- g. For **Configure propagate tags**, choose **Propagate tags from task definition** to have Amazon ECS add the tags associated with the task definition to your task. For more information, see [Tagging your resources \(p. 605\)](#).

Note

If you specify a tag with the same key in the **Tags** section, that tag overrides the tag that's propagated from the task definition.

14. 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.
 - 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**
15. For **Retry policy and dead-letter queue**: under **Retry policy**:
 - a. For **Maximum age of event**, enter a value between one minute (00:01) and 24 hours (24:00).
 - b. For **Retry attempts**, enter a number between 0 and 185.
16. 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 aren't 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 drop-down list.
 - 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*.

View your scheduled tasks

Your scheduled tasks can be viewed in the 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 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)

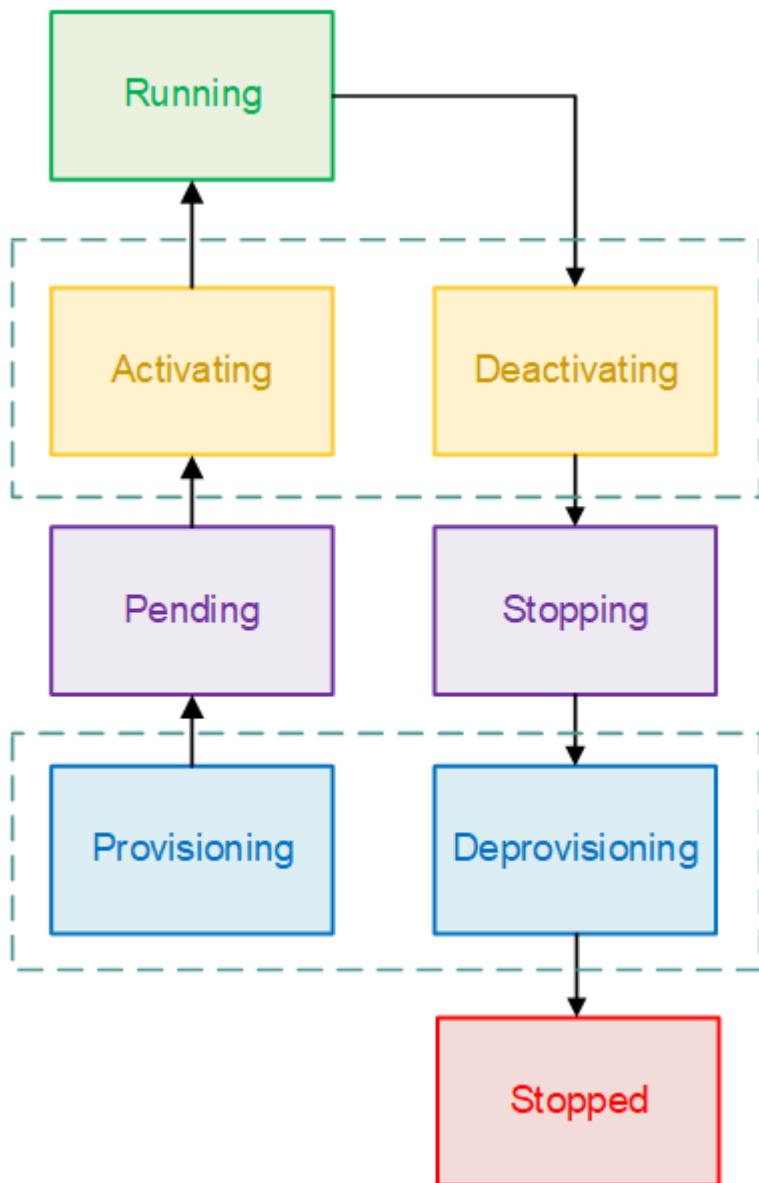
1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. Choose the cluster in which to edit your scheduled task.
3. On the Cluster: **cluster-name** page, choose **Scheduled Tasks**.
4. Select the box to the left of the schedule rule to edit, and choose **Edit**.
5. Edit the fields to update and choose **Update**.

Task lifecycle

When a task is started, either manually or as part of a service, it can pass through several states before it finishes on its own or is stopped manually. Some tasks are meant to run as batch jobs that naturally progress through from PENDING to RUNNING to STOPPED. Other tasks, which can be part of a service, are meant to continue running indefinitely, or to be scaled up and down as needed.

When task status changes are requested, such as stopping a task or updating the desired count of a service to scale it up or down, the Amazon ECS container agent tracks these changes as the last known status (`lastStatus`) of the task and the desired status (`desiredStatus`) of the task. Both the last known status and desired status of a task can be seen either in the console or by describing the task with the API or AWS CLI.

The flow chart below shows the task lifecycle flow.



Lifecycle states

The following are descriptions of each of the task lifecycle states.

PROVISIONING

Amazon ECS has to perform additional steps before the task is launched. For example, for tasks that use the `awsvpc` network mode, the elastic network interface needs to be provisioned.

PENDING

This is a transition state where Amazon ECS is waiting on the container agent to take further action.

ACTIVATING

Amazon ECS has to perform additional steps after the task is launched but before the task can transition to the `RUNNING` state. For example, for tasks that have service discovery configured, the

service discovery resources must be created. For tasks that are part of a service that's configured to use multiple Elastic Load Balancing target groups, the target group registration occurs during this state.

RUNNING

The task is successfully running.

DEACTIVATING

Amazon ECS has to perform additional steps before the task is stopped. For example, for tasks that are part of a service that's configured to use multiple Elastic Load Balancing target groups, the target group deregistration occurs during this state.

STOPPING

This is a transition state where Amazon ECS is waiting on the container agent to take further action.

DEPROVISIONING

Amazon ECS has to perform additional steps after the task has stopped but before the task transitions to the STOPPED state. For example, for tasks that use the awsvpc network mode, the elastic network interface needs to be detached and deleted.

STOPPED

The task has been successfully stopped.

Creating a scheduled task using the AWS CLI

This topic describes how to create a scheduled task using the AWS CLI. The scheduled task is created using the CloudWatch Events API. For more information, see [What is Amazon CloudWatch Events?](#) in the *Amazon CloudWatch Events User Guide*.

Complete the following prerequisites:

- Set up an AWS account and an `ecsEventsRole` associated with your account.
- Install and configure the AWS CLI version 2. For more information, see [Installing the AWS CLI version 2 and AWS Command Line Interface](#).
- A registered task definition. If you haven't yet created and registered a task definition, see [Getting started with the Amazon ECS console using Linux containers on AWS Fargate \(p. 30\)](#).
- An Amazon EC2 Linux instance running on your default ECS cluster. For instructions on how to create these resources, see [Getting started with the Amazon ECS console using Linux containers on AWS Fargate \(p. 30\)](#).

Before you verify the scheduling results, make sure that the cluster isn't running a service or task. From the ECS console, delete the cluster tasks and service before trying the example.

To create a scheduled task (AWS CLI)

1. Create the CloudWatch Events rule. This example creates a rule named `MyRule1` that's triggered every day at 12:00pm UTC. You can change the time so that it's more convenient for verifying the schedule results. The first time placeholder is minutes and the second placeholder is UTC hours. For other examples of rule expressions, see [Schedule Expressions for Rules](#) in the *Amazon CloudWatch Events User Guide*.

```
aws events put-rule \
    --schedule-expression "cron(0 12 * * ? *)"
    --name MyRule1
```

2. Add the details of your Amazon ECS cluster and task definition as a target for the CloudWatch Events rule. Specify the cluster and task definition using the full Amazon Resource Name (ARN). The launch type and network configuration must be defined either in the task definition or the `put-targets` command line. When using Fargate, the network configuration must be defined as `awsvpc`.

In this example, the target is defined as the `default` cluster in which to run a Fargate task based on the `first-run-task-definition:1` task definition. A count of one task is scheduled to run according to `MyRule1`. An `ecsEventsRole` IAM role is assigned to the target. The launch type is `FARGATE` and the network configuration is defined as `awsvpc` with a security groups and a public subnet. The command is run from the ECS instance in the default cluster. For more information about `put-targets`, see [put-targets](#). The cluster and task definition must already be created. Otherwise, you receive an error.

Create a local file named `scheduledtask.json` with the following contents:

```
[{  
    "Id": "1",  
    "Arn": "arn:aws:ecs:us-east-1:123456789012:cluster/default",  
    "RoleArn": "arn:aws:iam::123456789012:role/ecsEventsRole",  
    "EcsParameters": {  
        "TaskDefinitionArn": "arn:aws:ecs:us-east-1:123456789012:task-definition/first-run-task-definition:1",  
        "TaskCount": 1,  
        "LaunchType": "FARGATE",  
        "NetworkConfiguration": {  
            "awsvpcConfiguration": {  
                "Subnets": ["subnet1"],  
                "SecurityGroups": ["secgroup1"],  
                "AssignPublicIp": "ENABLED"  
            }  
        },  
        "PlatformVersion": "LATEST"  
    }  
}]
```

Use the following command to create the target:

```
aws events put-targets \  
  --rule "MyRule1" \  
  --targets file://scheduledtask.json
```

Amazon ECS services

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

In addition to maintaining the desired number of tasks in your service, you can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

Topics

- [Service scheduler concepts \(p. 531\)](#)
- [Additional service concepts \(p. 533\)](#)
- [Service definition parameters \(p. 534\)](#)
- [Creating an Amazon ECS service \(p. 546\)](#)
- [Updating a service \(p. 559\)](#)
- [Deleting a service \(p. 562\)](#)
- [Amazon ECS Deployment types \(p. 563\)](#)
- [Service load balancing \(p. 574\)](#)
- [Service auto scaling \(p. 591\)](#)
- [Service Discovery \(p. 599\)](#)
- [Service throttle logic \(p. 602\)](#)

Service scheduler concepts

The service scheduler is ideally suited for long running stateless services and applications. The service scheduler ensures that the scheduling strategy you specify is followed and reschedules tasks when a task fails (for example, if the underlying infrastructure fails for some reason). Task placement strategies and constraints can be used 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 the number of desired running tasks based on the scheduling strategy (also referred to as the *service type*) that the service uses.

The service scheduler includes logic that throttles how often tasks are restarted if they repeatedly fail to start. If a task is stopped without having entered a RUNNING state, determined by the task having a startedAt time stamp, the service scheduler starts to incrementally slow down the launch attempts and emits a service event message. This behavior prevents unnecessary resources from being used for failed tasks, giving you a chance to resolve the issue. After the service is updated, the service scheduler resumes normal behavior. For more information, see [Service throttle logic \(p. 602\)](#) and [Service event messages \(p. 819\)](#).

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 \(p. 533\)](#).
- **DAEMON**—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet

the placement constraints. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see [Daemon \(p. 532\)](#).

Note

Fargate tasks do not support the DAEMON scheduling strategy.

Daemon

The *daemon* scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints specified in your cluster. The service scheduler also evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies.

Amazon ECS reserves container instance compute resources including CPU, memory and network interfaces for the daemon tasks. When you launch a daemon service on a cluster with other replica services, Amazon ECS prioritizes the daemon task to be the first task to launch on the instances and the last task to stop. This strategy ensures that resources are not used by pending replica tasks and are available for the daemon tasks.

The daemon service scheduler does not place any tasks on instances that have a DRAINING status. If a container instance transitions to DRAINING, the daemon tasks on it are stopped. The service scheduler also monitors when new container instances are added to your cluster and adds the daemon tasks to them.

If a deployment configuration is specified, the maximum percent parameter must be 100. The default value for a daemon service for maximumPercent is 100%. The default value for a daemon service for minimumHealthyPercent is 0%.

A change the placement constraints for the daemon service requires a service restart for the changes to take effect. Amazon ECS dynamically updates the resources reserved on qualifying instances for the daemon task. For existing instances, the scheduler tries to place the task on the instance.

A change the task size or container resource reservation in the task-definition starts a deployment of the service. Amazon ECS picks up the updated CPU and memory reservations for the daemon, and then blocks that capacity for the daemon task.

If there are insufficient resources for either of the above cases, the following happens:

- The task placement fails.
- A CloudWatch event is generated.
- Amazon ECS continues to try and schedule the task on the instance by waiting for resources to become available.
- Amazon ECS frees up any reserved instances that no longer meet the placement constraint criteria and stops the corresponding daemon tasks.

The daemon scheduling strategy can be used in the following cases:

- Running application containers
- Running support containers for logging, monitoring and tracing tasks

Tasks using the Fargate launch type or the CODE_DEPLOY or EXTERNAL deployment controller types don't support the daemon scheduling strategy.

Note

The daemon service scheduler does not support the use of Classic Load Balancers.

When the service scheduler stops running tasks, it attempts to maintain balance across the Availability Zones in your cluster. The scheduler uses the following logic:

- If a placement strategy is defined, use that strategy to select which tasks to terminate. For example, if a service has an Availability Zone spread strategy defined, then a task is selected that leaves the remaining tasks with the best spread.
- If no placement strategy is defined, maintain balance across the Availability Zones in your cluster with the following logic:
 - Sort the valid container instances, giving priority to instances that have the largest number of running tasks for this service in their respective Availability Zone. For example, if zone A has one running service task and zones B and C each have two, container instances in either zone B or C are considered optimal for termination.
 - Stop the task on a container instance in an optimal Availability Zone (based on the previous steps), favoring container instances with the largest number of running tasks for this service.

Replica

The *replica* scheduling strategy places and maintains the desired number of tasks in your cluster.

When creating a service that runs tasks on Fargate, when the service scheduler launches new tasks or stops running tasks, it attempts to maintain balance across Availability Zones. There is no need to specify task placement strategies or restraints.

When creating a service that runs tasks on EC2 instances , you can optionally specify task placement strategies and constraints to customize task placement decisions. If no task placement strategies or constraints are specified, then by default the service scheduler will spread the tasks across Availability Zones. The service scheduler uses the following logic:

- Determine which of the container instances in your cluster can support your service's task definition (for example, they have the required CPU, memory, ports, and container instance attributes).
- Determine which container instances satisfy any placement constraints that are defined for the service.
- If there is a placement strategy defined, use that strategy to select an instance from the remaining candidates.
- If there is no placement strategy defined, balance tasks across the Availability Zones in your cluster with the following logic:
 - Sort the valid container instances, giving priority to instances that have the fewest number of running tasks for this service in their respective Availability Zone. For example, if zone A has one running service task and zones B and C each have zero, valid container instances in either zone B or C are considered optimal for placement.
 - Place the new service task on a valid container instance in an optimal Availability Zone (based on the previous steps), favoring container instances with the fewest number of running tasks for this service.

Additional service concepts

- You can optionally run your service behind a load balancer. For more information, see [Service load balancing \(p. 574\)](#).
- 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 \(p. 534\)](#).

- You can optionally configure your service to use Amazon ECS service discovery. Service discovery uses Amazon Route 53 auto naming APIs to manage DNS entries for your service's tasks, making them discoverable within your VPC. For more information, see [Service Discovery \(p. 599\)](#).
- When you delete a service, if there are still running tasks that require cleanup, the service status moves from ACTIVE to DRAINING, and the service is no longer visible in the console or in the ListServices API operation. After all tasks have transitioned to either STOPPING or STOPPED status, the service status moves from DRAINING to INACTIVE. Services in the DRAINING or INACTIVE status can still be viewed with the DescribeServices API operation. However, in the future, INACTIVE services may be cleaned up and purged from Amazon ECS record keeping, and DescribeServices calls on those services return a ServiceNotFoundException error.

Service definition parameters

A service definition defines how to run your Amazon ECS service. The following parameters can be specified in a service definition.

Launch type

`launchType`

Type: String

Valid values: EC2 | FARGATE | 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 \(p. 247\)](#).

If a `launchType` is specified, the `capacityProviderStrategy` parameter must be omitted.

Capacity provider strategy

`capacityProviderStrategy`

Type: Array of objects

Required: No

The capacity provider strategy to use for the service.

A capacity provider strategy consists of one or more capacity providers along with the base and weight to assign to them. A capacity provider must be associated with the cluster to be used in a capacity provider strategy. The PutClusterCapacityProviders API is used to associate a capacity provider with a cluster. Only capacity providers with an ACTIVE or UPDATING status can be used.

If a `capacityProviderStrategy` is specified, the `launchType` parameter must be omitted. If no `capacityProviderStrategy` or `launchType` is specified, the `defaultCapacityProviderStrategy` for the cluster is used.

If specifying a capacity provider that uses an Auto Scaling group, the capacity provider must already be created. New capacity providers can be created with the CreateCapacityProvider API operation.

To use a AWS Fargate capacity provider, specify either the FARGATE or FARGATE_SPOT capacity providers. The AWS Fargate capacity providers are available to all accounts and only need to be associated with a cluster to be used.

The PutClusterCapacityProviders API operation is used to update the list of available capacity providers for a cluster after the cluster is created.

`capacityProvider`

Type: String

Required: Yes

The short name or full ARN of the capacity provider.

`weight`

Type: Integer

Valid range: Integers between 0 and 1,000.

Required: No

The weight value designates the relative percentage of the total number of tasks launched that should use the specified capacity provider.

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

`base`

Type: Integer

Valid range: Integers between 0 and 100,000.

Required: No

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

Task definition

`taskDefinition`

Type: String

Required: No

The `family` and `revision` (`family:revision`) or full Amazon Resource Name (ARN) of the task definition to run in your service. If a `revision` is not specified, the latest ACTIVE revision of the specified family is used.

A task definition must be specified when using the rolling update (ECS) deployment controller.

Platform 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 \(p. 169\)](#).

Note

Platform versions are not specified for tasks using the EC2 launch type.

Cluster

`cluster`

Type: String

Required: No

The short name or full Amazon Resource Name (ARN) of the cluster on which to run your service. If you do not specify a cluster, the default cluster is assumed.

Service name

`serviceName`

Type: String

Required: Yes

The name of your service. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a Region or across multiple Regions.

Scheduling strategy

`schedulingStrategy`

Type: String

Valid values: `REPLICA` | `DAEMON`

Required: No

The scheduling strategy to use. If no scheduling strategy is specified, the `REPLICA` strategy is used. For more information, see [Service scheduler concepts \(p. 531\)](#).

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 \(p. 533\)](#).
- `DAEMON`—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see [Daemon \(p. 532\)](#).

Note

Fargate tasks do not support the `DAEMON` scheduling strategy.

Desired count

`desiredCount`

Type: Integer

Required: No

The number of instantiations of the specified task definition to place and keep running on your cluster.

This parameter is required if the `REPLICA` scheduling strategy is used. If the service uses the `DAEMON` scheduling strategy, this parameter is optional.

Deployment configuration

`deploymentConfiguration`

Type: Object

Required: No

Optional deployment parameters that control how many tasks run during the deployment and the ordering of stopping and starting tasks.

`maximumPercent`

Type: Integer

Required: No

If a service is using the rolling update (`ECS`) deployment type, the `maximumPercent` parameter represents an upper limit on the number of your service's tasks that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the `desiredCount` (rounded down to the nearest integer). This parameter enables you to define the deployment batch size. For example, if your service is using the `REPLICA` service scheduler and has a `desiredCount` of four tasks and a `maximumPercent` value of 200%, the scheduler may start four new tasks before stopping the four older tasks (provided that the cluster resources required to do this are available). The default `maximumPercent` value for a service using the `REPLICA` service scheduler is 200%.

If your service is using the `DAEMON` service scheduler type, the `maximumPercent` should remain at 100%, which is the default value.

The maximum number of tasks during a deployment is the `desiredCount` multiplied by the `maximumPercent/100`, rounded down to the nearest integer value.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and tasks that use the `EC2` launch type, the **maximum percent** value is set to the default value and is used to define the upper limit on the number of the tasks in the service that remain in the `RUNNING` state while the container instances are in the `DRAINING` state. If the tasks in the service use the `Fargate` launch type, the maximum percent value is not used, although it is returned when describing your service.

`minimumHealthyPercent`

Type: Integer

Required: No

If a service is using the rolling update (`ECS`) deployment type, the `minimumHealthyPercent` represents a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment, as a percentage of the `desiredCount` (rounded up to the nearest integer). This parameter enables you to deploy without using additional cluster capacity. For example, if your service has a `desiredCount` of four tasks and a `minimumHealthyPercent` of 50%, the service scheduler may stop two existing tasks to free up cluster capacity before starting two new tasks.

For services that *do not* use a load balancer, the following should be noted:

- A service is considered healthy if all essential containers within the tasks in the service pass their health checks.
- If a task has no essential containers with a health check defined, the service scheduler will wait for 40 seconds after a task reaches a `RUNNING` state before the task is counted towards the minimum healthy percent total.
- If a task has one or more essential containers with a health check defined, the service scheduler will wait for the task to reach a healthy status before counting it towards the minimum healthy percent total. A task is considered healthy when all essential containers within the task have passed their health checks. The amount of time the service scheduler can wait for is determined by the container health check settings. For more information, see [Health check \(p. 218\)](#).

For services that *do* use a load balancer, the following should be noted:

- If a task has no essential containers with a health check defined, the service scheduler will wait for the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.
- If a task has an essential container with a health check defined, the service scheduler will wait for both the task to reach a healthy status and the load balancer target group health check to return a healthy status before counting the task towards the minimum healthy percent total.

The default value for a replica service for `minimumHealthyPercent` is 100%. The default `minimumHealthyPercent` value for a service using the DAEMON service schedule is 0% for the AWS CLI, the AWS SDKs, and the APIs and 50% for the AWS Management Console.

The minimum number of healthy tasks during a deployment is the `desiredCount` multiplied by the `minimumHealthyPercent/100`, rounded up to the nearest integer value.

If a service is using either the blue/green (`CODE_DEPLOY`) or `EXTERNAL` deployment types and 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 \(p. 563\)](#).

`type`

Type: String

Valid values: `ECS` | `CODE_DEPLOY` | `EXTERNAL`

Required: yes

The deployment controller type to use. There are three deployment controller types available:
`ECS`

The rolling update (`ECS`) deployment type involves replacing the current running version of the container with the latest version. The number of containers Amazon ECS adds or removes from the service during a rolling update is controlled by adjusting the minimum and maximum number of healthy tasks allowed during a service deployment, as specified in the [deploymentConfiguration](#).

`CODE_DEPLOY`

The blue/green (`CODE_DEPLOY`) deployment type uses the blue/green deployment model powered by CodeDeploy, which allows you to verify a new deployment of a service before sending production traffic to it.

`EXTERNAL`

The external deployment type enables you to use any third party deployment controller for full control over the deployment process for an Amazon ECS service.

Task placement

`placementConstraints`

Type: Array of objects

Required: No

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task (this limit includes constraints in the task definition and those specified at run time). If you are using the Fargate launch type, task placement constraints are not supported.

type

Type: String

Required: No

The type of constraint. Use `distinctInstance` to ensure that each task in a particular group is running on a different container instance. Use `memberOf` to restrict the selection to a group of valid candidates. The value `distinctInstance` is not supported in task definitions.

expression

Type: String

Required: No

A cluster query language expression to apply to the constraint. Note you cannot specify an expression if the constraint type is `distinctInstance`. For more information, see [Cluster query language \(p. 520\)](#).

placementStrategy

Type: Array of objects

Required: No

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

type

Type: String

Valid values: `random` | `spread` | `binpack`

Required: No

The type of placement strategy. The `random` placement strategy randomly places tasks on available candidates. The `spread` placement strategy spreads placement across available candidates evenly based on the `field` parameter. The `binpack` strategy places tasks on available candidates that have the least available amount of the resource that is specified with the `field` parameter. For example, if you binpack on `memory`, a task is placed on the instance with the least amount of remaining memory (but still enough to run the task).

field

Type: String

Required: No

The field to apply the placement strategy against. For the `spread` placement strategy, valid values are `instanceId` (or `host`, which has the same effect), or any platform or custom attribute that is applied to a container instance, such as `attribute:ecs.availability-zone`. For the `binpack` placement strategy, valid values are `cpu` and `memory`. For the `random` placement strategy, this field is not used.

Tags

tags

Type: Array of objects

Required: No

The metadata that you apply to the service to help you categorize and organize them. Each tag consists of a key and an optional value, both of which you define. When a service is deleted, the tags are deleted as well. A maximum of 50 tags can be applied to the service. For more information, see [Tagging your Amazon ECS resources \(p. 604\)](#).

`key`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

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

`value`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

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

`enableECSManagedTags`

Type: Boolean

Valid values: `true` | `false`

Required: No

Specifies whether to use Amazon ECS managed tags for the tasks in the service. If no value is specified, the default value is `false`. For more information, see [Tagging your resources for billing \(p. 606\)](#).

`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 is not supported for other network modes. If using the Fargate launch type, the `awsvpc` network mode is required. For more information, see [Amazon ECS task networking \(p. 278\)](#).

`awsvpcConfiguration`

Type: Object

Required: No

An object representing the subnets and security groups for a task or service.

`subnets`

Type: Array of strings

Required: Yes

The subnets associated with the task or service. There is a limit of 16 subnets that can be specified per `awsvpcConfiguration`.

`securityGroups`

Type: Array of strings

Required: No

The security groups associated with the task or service. If you do not specify a security group, the default security group for the VPC is used. There is a limit of 5 security groups that can be specified per `awsvpcConfiguration`.

`assignPublicIP`

Type: String

Valid values: `ENABLED` | `DISABLED`

Required: No

Whether the task's elastic network interface receives a public IP address. If no value is specified, the default value of `DISABLED` is used.

`healthCheckGracePeriodSeconds`

Type: Integer

Required: No

The period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks, container health checks, and Route 53 health checks after a task enters a `RUNNING` state. This is only valid if your service is configured to use a load balancer. If your service has a load balancer defined and you do not specify a health check grace period value, the default value of 0 is used.

If your service's tasks take a while to start and respond to health checks, you can specify a health check grace period of up to 2,147,483,647 seconds during which the ECS service scheduler ignores the health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up.

`loadBalancers`

Type: Array of objects

Required: No

A load balancer object representing the load balancers to use with your service. For services that use an Application Load Balancer or Network Load Balancer, there is a limit of five target groups you can attach to a service.

After you create a service, the load balancer name or target group ARN, container name, and container port specified in the service definition are immutable.

For Classic Load Balancers, this object must contain the load balancer name, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance is registered with the load balancer specified here.

For Application Load Balancers and Network Load Balancers, this object must contain the load balancer target group ARN, the container name (as it appears in a container definition), and the container port to access from the load balancer. When a task from this service is placed on a container instance, the container instance and port combination is registered as a target in the target group specified here.

targetGroupArn

Type: String

Required: No

The full ARN of the Elastic Load Balancing target group associated with a service.

A target group ARN is only specified when using an Application Load Balancer or Network Load Balancer. If you are using a Classic Load Balancer the target group ARN should be omitted.

loadBalancerName

Type: String

Required: No

The name of the load balancer to associate with the service.

A load balancer name is only specified when using a Classic Load Balancer. If you are using an Application Load Balancer or a Network Load Balancer the load balancer name parameter should be omitted.

containerName

Type: String

Required: No

The name of the container (as it appears in a container definition) to associate with the load balancer.

containerPort

Type: Integer

Required: No

The port on the container to associate with the load balancer. This port must correspond to a `containerPort` in the task definition used by tasks in the service. For tasks that use the EC2 launch type, the container instance must allow ingress traffic on the `hostPort` of the port mapping.

role

Type: String

Required: No

The short name or full ARN of the IAM role that allows Amazon ECS to make calls to your load balancer on your behalf. This parameter is only permitted if you are using a load balancer with 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 \(p. 685\)](#).

`serviceRegistries`

Type: Array of objects

Required: No

The details of the service discovery configuration for your service. For more information, see [Service Discovery \(p. 599\)](#).

`registryArn`

Type: String

Required: No

The ARN of the service registry. The currently supported service registry is AWS Cloud Map. For more information, see [Working with Services](#) in the *AWS Cloud Map Developer Guide*.

`port`

Type: Integer

Required: No

The port value used if your service discovery service specified an SRV record. This field is required if both the awsvpc network mode and SRV records are used.

`containerName`

Type: String

Required: No

The container name value, already specified in the task definition, to be used for your service discovery service. If the task definition that your service task specifies uses the bridge or host network mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition that your service task specifies uses the awsvpc network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

`containerPort`

Type: Integer

Required: No

The port value, already specified in the task definition, to be used for your service discovery service. If the task definition your service task specifies uses the bridge or host network

mode, you must specify a `containerName` and `containerPort` combination from the task definition. If the task definition your service task specifies uses the `awsvpc` network mode and a type SRV DNS record is used, you must specify either a `containerName` and `containerPort` combination or a `port` value, but not both.

Client token

`clientToken`

Type: String

Required: No

Unique, case-sensitive identifier you provide to ensure the idempotency of the request. Up to 32 ASCII characters are allowed.

Service definition template

The following shows the JSON representation of an Amazon ECS service definition.

```
{  
    "cluster": "",  
    "serviceName": "",  
    "taskDefinition": "",  
    "loadBalancers": [  
        {  
            "targetGroupArn": "",  
            "loadBalancerName": "",  
            "containerName": "",  
            "containerPort": 0  
        }  
    ],  
    "serviceRegistries": [  
        {  
            "registryArn": "",  
            "port": 0,  
            "containerName": "",  
            "containerPort": 0  
        }  
    ],  
    "desiredCount": 0,  
    "clientToken": "",  
    "launchType": "FARGATE",  
    "capacityProviderStrategy": [  
        {  
            "capacityProvider": "",  
            "weight": 0,  
            "base": 0  
        }  
    ],  
    "platformVersion": "",  
    "platformFamily": "",  
    "role": "",  
    "deploymentConfiguration": {  
        "maximumPercent": 0,  
        "minimumHealthyPercent": 0  
    },  
    "placementConstraints": [  
        {  
            "type": "distinctInstance",  
            "value": ""  
        }  
    ]  
}
```

```
        "expression": ""
    },
    "placementStrategy": [
        {
            "type": "spread",
            "field": ""
        }
    ],
    "networkConfiguration": {
        "awsVpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "ENABLED"
        }
    },
    "healthCheckGracePeriodSeconds": 0,
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "tags": [
        {
            "key": "",
            "value": ""
        }
    ],
    "enableECSManagedTags": true,
    "propagateTags": "SERVICE"
}
```

You can create this service definition template using the following AWS CLI command.

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

Creating an Amazon ECS service

When you create an Amazon ECS service, you specify the basic parameters that define what makes up your service and how it should behave. These parameters create a service definition. For more information, see [Service definition parameters \(p. 534\)](#).

For services 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 \(p. 574\)](#).

Note

When using a load balancer with services hosted on Amazon EC2 instances, you should 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 to ensure that traffic can reach any containers that use dynamically assigned ports.

We provide walkthroughs for creating an Amazon ECS service using the AWS Management Console in the following pages.

Topics

- [Creating a service using the new console \(p. 547\)](#)
- [Creating a service using the classic console \(p. 548\)](#)

Creating a service using the new console

Important

If you are creating a Windows service for the Fargate launch type, you must use the classic console. For more information, see [Creating a service using the classic console \(p. 548\)](#).

You can create an Amazon ECS service using the new Amazon ECS 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 using the new console

1. Open the Amazon ECS 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 old 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.

Creating a service using the classic 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 \(p. 547\)](#).

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

The Amazon ECS console provides a create service wizard which 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 \(p. 549\)](#)
- [Step 2: Configure a network \(p. 551\)](#)
- [Step 3: Configuring your service to use a load balancer \(p. 552\)](#)
- [Step 4: Configuring your service to use Service Discovery \(p. 556\)](#)
- [Step 5: Configuring your service to use Service Auto Scaling \(p. 557\)](#)
- [Step 6: Review and create your service \(p. 559\)](#)

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 \(p. 547\)](#).

All services require some basic configuration parameters that define the service, such as the task definition to use, which cluster the service should run on, how many tasks should be placed for the service, and so on. This is called the *service definition*. For more information about the parameters defined in a service definition, see [Service definition parameters \(p. 534\)](#).

This procedure covers creating a service with the basic service definition parameters that are required. After you have configured these parameters, you can create your service or move on to the procedures for optional service definition configuration, such as configuring your service to use a load balancer.

Note

If your cluster is configured with a default capacity provider strategy, you will only be able to create a service using the default capacity provider strategy when using the console. Likewise, if no default capacity provider is defined, you will only be able to use a launch type when creating a service using the console. It is not currently possible to have a mixed strategy using both capacity providers and launch types in the console.

To configure the basic service definition parameters

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the navigation bar, select the Region that your cluster is in.
3. In the navigation pane, choose **Task Definitions** and select the task definition from which to create your service.
4. On the **Task Definition name** page, select the revision of the task definition from which to create your service.
5. Review the task definition, and choose **Actions, Create Service**.
6. On the **Configure service** page, complete the following steps.
 - a. Choose either a capacity provider strategy or a launch type.
 - To use a **Capacity provider strategy**, choose **Switch to capacity provider strategy** and then choose whether your service should use the default capacity provider strategy defined for the cluster or a custom capacity provider strategy. A capacity provider must already be associated with the cluster in order to be used in a custom capacity provider strategy. For more information, see [Amazon ECS capacity providers \(p. 178\)](#).
 - 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 \(p. 247\)](#).
 - 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 in which to create your service.
- e. **Service name:** Type a unique name for your service.
- f. **Service type:** Select a scheduling strategy for your service. For more information, see [Service scheduler concepts \(p. 531\)](#).

- g. **Number of tasks:** If you chose the `REPLICA` service type, type the number of tasks to launch and maintain on your cluster.

Note

If your launch type is `EC2`, and your task definition uses static host port mappings on your container instances, then you need at least one container instance with the specified port available in your cluster for each task in your service. This restriction does not apply if your task definition uses dynamic host port mappings with the `bridge` network mode. For more information, see [portMappings \(p. 216\)](#).

- h. If you are using the **Rolling update** deployment type, fill out the following deployment configuration parameters. For more information on how these parameters are used, see [Deployment configuration \(p. 537\)](#).
- **Minimum healthy percent:** Specify a lower limit on the number of your service's tasks that must remain in the `RUNNING` state during a deployment, as a percentage of the service's desired number of tasks (rounded up to the nearest integer).
 - **Maximum percent:** Specify an upper limit on the number of your service's tasks that are allowed in the `RUNNING` or `PENDING` state during a deployment, as a percentage of the service's desired number of tasks (rounded down to the nearest integer).
7. For **Deployment circuit breaker**, choose the deployment circuit breaker logic. For more information, see [the section called "Using the deployment circuit breaker" \(p. 564\)](#).
8. On the **Deployments** page, complete the following steps.
- a. For **Deployment type**, choose whether your service should use a rolling update deployment or a blue/green deployment using AWS CodeDeploy. For more information, see [Amazon ECS Deployment types \(p. 563\)](#).
 - 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 \(p. 565\)](#)
 - ii. For **Service role for CodeDeploy** choose the IAM service role for AWS CodeDeploy. For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 704\)](#)
9. (Optional) If you selected the `EC2` launch type and the `REPLICA` service type, for **Task Placement**, you can specify how tasks are placed using task placement strategies and constraints. Choose from the following options. For more information, see [Amazon ECS task placement \(p. 513\)](#).
- **AZ Balanced Spread** - Distribute tasks across Availability Zones and across container instances in the Availability Zone.
 - **AZ Balanced BinPack** - Distribute tasks across Availability Zones and across container instances with the least available memory.
 - **BinPack** - Distribute tasks based on the least available amount of CPU or memory.
 - **One Task Per Host** - Place, at most, one task from the service on each container instance.
 - **Custom** - Define your own task placement strategy. See [Amazon ECS task placement \(p. 513\)](#) for examples.
10. 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.

11. 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](#).
12. Choose **Next step** and navigate to [Step 2: Configure a network \(p. 551\)](#).

Step 2: Configure a network

Important

Amazon ECS has provided a new console experience for creating a service. For more information, see [Creating a service using the new console \(p. 547\)](#).

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 \(p. 552\)](#).

For tasks hosted on Amazon EC2 instances, the `awsvpc` network mode doesn't provide task ENIs with public IP addresses. To access the internet, tasks hosted on Amazon EC2 instances can be launched in a private subnet that is configured to use a NAT gateway. For more information, see [NAT Gateways](#) in the *Amazon VPC User Guide*. Inbound network access must be from within the VPC using the private IP address or DNS hostname, or routed through a load balancer from within the VPC. Tasks launched within public subnets do not have internet access.

To configure VPC and security group settings for your service

1. If you have not done so already, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 549\)](#).
2. For **Cluster VPC**, if you are hosting tasks on Amazon EC2 instances, choose the VPC in which your instances reside. If you're hosting tasks on Fargate, select the VPC that the Amazon ECS on Fargate tasks should use. Ensure that the VPC you choose is not configured to require dedicated hardware tenancy, as that isn't supported by Fargate.
3. For **Subnets**, choose the available subnets for your service task placement.
4. For **Security groups**, a security group has been created for your service's tasks, which allows HTTP traffic from the internet (`0.0.0.0/0`). To edit the name or the rules of this security group, or to choose an existing security group, choose **Edit** and then modify your security group settings.
5. For **Auto-assign Public IP**, choose whether to have your tasks receive a public IP address. For tasks on Fargate, in order for the task to pull the container image it must either use a public subnet and be assigned a public IP address or a private subnet that has a route to the internet or a NAT gateway that can route requests to the internet.
6. If you are configuring your service to use a load balancer or if you are using the blue/green deployment type, continue to [Step 3: Configuring your service to use a load balancer \(p. 552\)](#). If you are not configuring your service to use a load balancer, you can choose **None** as the load balancer type and move on to the next section, [Step 5: Configuring your service to use Service Auto Scaling \(p. 557\)](#).

Step 3: Configuring your service to use a load balancer

Important

Amazon ECS has provided a new console experience for creating a service. For more information, see [Creating a service using the new console \(p. 547\)](#).

Services can be configured to use a load balancer to distribute incoming traffic to the tasks in your service. If your service is using the rolling update deployment type, this is optional. If your service is using the blue/green deployment type, then it is required to use either an Application Load Balancer or Network Load Balancer.

If you are not configuring your service to use a load balancer, you can choose **None** as the load balancer type and move on to the next section, [Step 4: Configuring your service to use Service Discovery \(p. 556\)](#).

If you have an available Elastic Load Balancing load balancer configured, you can attach it to your service with the following procedures, or you can configure a new load balancer. For more information, see [Creating a load balancer \(p. 579\)](#).

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 \(p. 552\)](#)
- [Configuring a load balancer for the blue/green deployment type \(p. 554\)](#)

Configuring a load balancer for the rolling update deployment type

If your service's tasks take a while to start and respond to Elastic Load Balancing health checks, you can specify a health check grace period of up to 2,147,483,647 seconds. During that time, the service scheduler ignores health check status. This grace period can prevent the service scheduler from marking tasks as unhealthy and stopping them before they have time to come up. This is only valid if your service is configured to use a load balancer.

To configure a health check grace period

1. If you have not done so already, follow the basic service configuration procedures in [Step 1: Configuring basic service parameters \(p. 549\)](#).
2. For **Health check grace period**: Enter the period of time, in seconds, that the Amazon ECS service scheduler should ignore unhealthy Elastic Load Balancing target health checks after a task has first started.

To configure your service to use a load balancer, you must choose the load balancer type to use with your service.

To choose a load balancer type

1. If you have not done so already, follow the basic service creation procedures in [Step 1: Configuring basic service parameters \(p. 549\)](#).
2. For **Load balancer type**, choose the load balancer type to use with your service:

Application Load Balancer

Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.

Network Load Balancer

Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing.

Classic Load Balancer

Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of the advanced features available to them.

3. For **Select IAM role for service**, choose **Create new role** to create 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 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 \(p. 553\)](#). If you've chosen a Network Load Balancer, follow the steps in [To configure a Network Load Balancer \(p. 554\)](#). If you've chosen a Classic Load Balancer, follow the steps in [To configure a Classic Load Balancer \(p. 554\)](#).

To configure an Application Load Balancer

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose **Add to load balancer**.
2. For **Listener port**, choose the listener port and protocol of the listener that you created in [Creating an Application Load Balancer \(p. 580\)](#) (if applicable), or choose **create new** to create a new listener and then enter a port number and choose a port protocol for **Listener protocol**.
3. For **Target group name**, choose the target group that you created in [Creating an Application Load Balancer \(p. 580\)](#) (if applicable), or choose **create new** to create a new target group.

Important

If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), your target group must use `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

4. (Optional) If you chose to create a new target group, complete the following fields as follows:
 - For **Target group name**, a default name is provided for you.
 - For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.
 - For **Path pattern**, if your listener does not have any existing rules, the default path pattern `(/)` is used. If your listener already has a default rule, then you must enter a path pattern that matches traffic that you want to have sent to your service's target group. For example, if your service is a web application called `web-app`, and you want traffic that matches `http://my-elb-url/web-app` to route to your service, then you would enter `/web-app*` as your path pattern. For more information, see [ListenerRules](#) in the *User Guide for Application Load Balancers*.
 - For **Health check path**, enter the path to which the load balancer should send health check pings.
5. When you are finished configuring your Application Load Balancer, choose **Next step**.

To configure a Network Load Balancer

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose **Add to load balancer**.
2. For **Listener port**, choose the listener port and protocol of the listener that you created in [Creating a Network Load Balancer \(p. 584\)](#) (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 \(p. 584\)](#) (if applicable), or choose **Create new** to create a new target group.

Important

If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), your target group must use `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

4. (Optional) If you chose to create a new target group, complete the following fields as follows:
 - For **Target group name**, a default name is provided for you.
 - For **Target group protocol**, enter the protocol to use for routing traffic to your tasks.
 - For **Health check path**, enter the path to which the load balancer should send health check pings.
5. When you are finished configuring your Network Load Balancer, choose **Next Step**.

To configure a Classic Load Balancer

1. The **Health check port**, **Health check protocol**, and **Health check path** fields are all pre-populated with the values you configured in [Creating a Classic Load Balancer \(p. 585\)](#) (if applicable). You can update these settings in the Amazon EC2 console.
2. For **Container for ELB health check**, choose the container to send health checks.
3. When you are finished configuring your Classic Load Balancer, choose **Next step**.

Configuring a load balancer for the blue/green deployment type

To configure your service that uses the blue/green deployment type to use a load balancer, you must use either an Application Load Balancer or a Network Load Balancer.

To choose a load balancer type

1. If you have not done so already, follow the basic service creation procedures in [Step 1: Configuring basic service parameters \(p. 549\)](#).
2. For **Load balancer type**, choose the load balancer type to use with your service:

Application Load Balancer

Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.

Network Load Balancer

Allows containers to use dynamic host port mapping, which enables you to place multiple tasks using the same port on a single container instance. Multiple services can use the same listener port on a single load balancer with rule-based routing.

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of the advanced features available to them.

3. For **Load balancer name**, choose the name of the load balancer to use with your service. Only load balancers that correspond to the load balancer type you selected earlier are visible here.
4. The next step depends on the load balancer type for your service. If you've chosen an Application Load Balancer, follow the steps in [To configure an Application Load Balancer \(p. 553\)](#). If you've chosen a Network Load Balancer, follow the steps in [To configure a Network Load Balancer \(p. 554\)](#).

To configure an Application Load Balancer for the blue/green deployment type

1. For **Container to load balance**, choose the container and port combination from your task definition that your load balancer should distribute traffic to, and choose **Add to load balancer**.
2. For **Production listener port**, choose the listener port and protocol of the listener that you created in [Creating an Application Load Balancer \(p. 580\)](#) (if applicable), or choose **create new** to create a new listener and then enter a port number and choose a port protocol for **Production listener protocol**.
3. (Optional) Select **Test listener** if you want to configure a listener port and protocol on your load balancer to test updates to your service before routing traffic to your new taskset. Complete the following step:
 - For **Test listener port**, choose the listener port and protocol of the listener that you want to test traffic over, or choose **create new** to create a new test listener and then enter a port number and choose a port protocol in **Test listener protocol**.
4. For blue/green deployments, two target groups are required. Each target group binds to a separate taskset in the deployment. Complete the following steps:
 - a. For **Target group 1 name**, choose the target group that you created in [Creating an Application Load Balancer \(p. 580\)](#) (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 \(p. 556\)](#).

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 \(p. 584\)](#) (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 \(p. 584\)](#) (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 \(p. 556\)](#).

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 \(p. 599\)](#).

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 \(p. 557\)](#).

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 \(p. 549\)](#).
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 \(p. 557\)](#).

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 \(p. 593\)](#) (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 \(p. 597\)](#)—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 \(p. 591\)](#).

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 \(p. 549\)](#).
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.

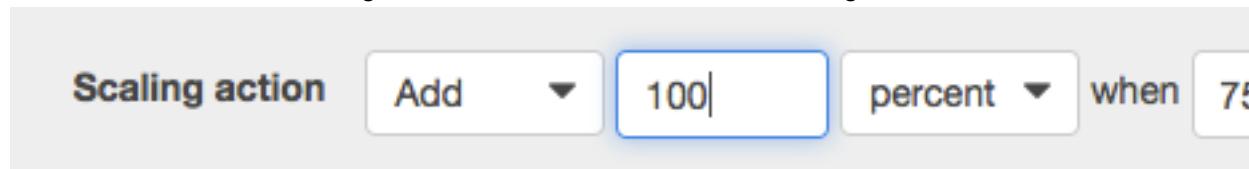
To configure step scaling policies for your service

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 \(p. 591\)](#).
 - c. For **Alarm threshold**, enter the following information to configure your alarm:
 - Choose the CloudWatch statistic for your alarm (the default value of **Average** works in many cases). For more information, see [Statistics](#) in the *Amazon CloudWatch User Guide*.
 - Choose the comparison operator for your alarm and enter the value that the comparison operator checks against (for example, `>` and 75).
 - Enter the number of consecutive periods before the alarm is triggered and the period length. For example, two consecutive periods of 5 minutes would take 10 minutes before the alarm triggered. Because your Amazon ECS tasks can scale up and down quickly, consider using a low number of consecutive periods and a short period duration to react to alarms as soon as possible.
 - d. Choose **Save**.
5. For **Scaling action**, enter the following information to configure how your service responds to the alarm:

- Choose whether to add to, subtract from, or set a specific desired count for your service.
- If you chose to add or subtract tasks, enter the number of tasks (or percent of existing tasks) to add or subtract when the scaling action is triggered. If you chose to set the desired count, enter the desired count that your service should be set to when the scaling action is triggered.
- (Optional) If you chose to add or subtract tasks, choose whether the previous value is used as an integer or a percent value of the existing desired count.
- Enter the lower boundary of your step scaling adjustment. By default, for your first scaling action, this value is the metric amount where your alarm is triggered. For example, the following scaling action adds 100% of the existing desired count when the CPU utilization is greater than 75%.



6. (Optional) You can repeat [Step 5 \(p. 558\)](#) 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 **Coldown period**, enter the number of seconds between scaling actions.
9. Repeat [Step 1 \(p. 558\)](#) through [Step 8 \(p. 559\)](#) for the **Scale in** policy and choose **Save**.
10. Choose **Next step** to proceed and navigate to [Step 6: Review and create your service \(p. 559\)](#).

Step 6: Review and create your service

After you have configured your basic service definition parameters and optionally configured your service's networking, load balancer, service discovery, and automatic scaling, you can review your configuration. Then, choose **Create Service** to finish creating your service.

Note

After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.

Updating a service

You can update an existing service to change some of the service configuration parameters, such as the number of tasks that are maintained by a service, which task definition is used by the tasks, or if your tasks are using the Fargate launch type, you can change the platform version your service uses. 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 new console \(p. 560\)](#)
- [Updating a service using the old console \(p. 561\)](#)

Updating a service using the new console

You can update an Amazon ECS service using the new Amazon ECS 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.

To create a service using the new console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/v2>.
2. On the **Clusters** page, select the cluster to create the service in.
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).
- 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**.

Updating a service using the old 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 \(p. 560\)](#).

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 \(p. 557\)](#).
 - c. Complete the steps in that section and then return.
12. Choose **Update Service** to finish and update your service.

Deleting a service

You can delete an Amazon ECS service using the console. Before deletion, the service is automatically scaled down to zero. If you have a load balancer or service discovery resources associated with the service, they are not affected by the service deletion. To delete your Elastic Load Balancing resources, see

one of the following topics, depending on your load balancer type: [Delete an Application Load Balancer](#) or [Delete a Network Load Balancer](#). To delete your service discovery resources, follow the procedure below.

To delete an Amazon ECS service

Use the following procedure to delete an Amazon ECS service.

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the navigation bar, select the Region that your cluster is in.
3. In the navigation pane, choose **Clusters** and select the name of the cluster in which your service resides.
4. On the **Cluster : name** page, choose **Services**.
5. Check the box to the left of the service to update and choose **Delete**.
6. Confirm the service deletion by entering the text phrase and choose **Delete**.

To delete the service discovery resources (AWS CLI)

To delete the remaining service discovery resources, you can use the AWS CLI to delete the service discovery service and service discovery namespace.

1. Ensure that the latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see [Installing the AWS Command Line Interface](#).
2. Retrieve the ID of the service discovery service to delete.

```
aws servicediscovery list-services --region <region_name>
```

Note

If no service discovery service is returned, continue to step 4.

3. Using the service discovery service ID from the previous output, delete the service.

```
aws servicediscovery delete-service --id <service_discovery_service_id> --region <region_name>
```

4. Retrieve the ID of the service discovery namespace to delete.

```
aws servicediscovery list-namespaces --region <region_name>
```

5. Using the service discovery namespace ID from the previous output, delete the namespace.

```
aws servicediscovery delete-namespace --id <service_discovery_namespace_id> --region <region_name>
```

Amazon ECS Deployment types

An Amazon ECS deployment type determines the deployment strategy that your service uses. There are three deployment types: rolling update, blue/green, and external.

Topics

- [Rolling update \(p. 564\)](#)
- [Blue/Green deployment with CodeDeploy \(p. 565\)](#)
- [External deployment \(p. 569\)](#)

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 is four then 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. \(p. 823\)](#).

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 \(p. 642\)](#).

To create a new Amazon ECS service that uses the rolling update deployment type, see [Creating an Amazon ECS service \(p. 546\)](#).

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 \(p. 642\)](#).
- 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.

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.ECSLinear10PercentEvery10Seconds	Shifts 10 percent of traffic every minute until all traffic is shifted.
CodeDeployDefault.ECSLinear10PercentEvery30Seconds	Shifts 10 percent of traffic every three minutes until all traffic is shifted.
CodeDeployDefault.ECSCanary10percent5M	Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed five minutes later.
CodeDeployDefault.ECSCanary10percent15M	Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed 15 minutes later.
CodeDeployDefault.ECSAllAtOnce	Shifts all traffic to the updated Amazon ECS container at once.

Revision

A revision is the CodeDeploy application specification file (AppSpec file). In the AppSpec file, you specify the full ARN of the task definition and the container and port of your replacement task set where traffic is to be routed when a new deployment is created. The container name must be one of the container names referenced in your task definition. If the network configuration or platform version has been updated in the service definition, you must also specify those details in the AppSpec file. You can also specify the Lambda functions to run during the deployment lifecycle events. The Lambda functions allow you to run tests and return metrics during the deployment. For more information, see [AppSpec File Reference](#) in the *AWS CodeDeploy User Guide*.

Blue/Green Deployment Considerations

Consider the following when using the blue/green deployment type:

- When an Amazon ECS service using the blue/green deployment type is initially created, an Amazon ECS task set is created.
- You must configure the service to use either an Application Load Balancer or Network Load Balancer. Classic Load Balancers aren't supported. The following are the load balancer requirements:
 - You must add a production listener to the load balancer, which is used to route production traffic.

- An optional test listener can be added to the load balancer, which is used to route test traffic. If you specify a test listener, CodeDeploy routes your test traffic to the replacement task set during a deployment.
- Both the production and test listeners must belong to the same load balancer.
- You must define a target group for the load balancer. The target group routes traffic to the original task set in a service through the production listener.
- When a Network Load Balancer is used, only the `CodeDeployDefault.ECSAllAtOnce` deployment configuration is supported.
- For services configured to use service auto scaling and the blue/green deployment type, auto scaling is not blocked during a deployment but the deployment may fail under some circumstances. The following describes this behavior in more detail.
 - If a service is scaling and a deployment starts, the green task set is created and CodeDeploy will wait up to an hour for the green task set to reach steady state and won't shift any traffic until it does.
 - If a service is in the process of a blue/green deployment and a scaling event occurs, traffic will continue to shift for 5 minutes. If the service doesn't reach steady state within 5 minutes, CodeDeploy will stop the deployment and mark it as failed.
- Tasks using the Fargate launch type or the `CODE_DEPLOY` deployment controller types don't support the `DAEMON` scheduling strategy.
- When you initially create a CodeDeploy application and deployment group, you must specify the following:
 - You must define two target groups for the load balancer. One target group should be the initial target group defined for the load balancer when the Amazon ECS service was created. The second target group's only requirement is that it can't be associated with a different load balancer than the one the service uses.
 - When you create a CodeDeploy deployment for an Amazon ECS service, CodeDeploy creates a *replacement task set* (or *green task set*) in the deployment. If you added a test listener to the load balancer, CodeDeploy routes your test traffic to the replacement task set. This is when you can run any validation tests. Then CodeDeploy reroutes the production traffic from the original task set to the replacement task set according to the traffic rerouting settings for the deployment group.

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 \(p. 546\)](#).

To update an existing Amazon ECS service that is using the blue/green deployment type, see [Updating a service \(p. 559\)](#).

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	<code>AppECS-<cluster[:47]>-<service[:47]></code>
Deployment group name	<code>DgpECS-<cluster[:47]>-<service[:47]></code>
Deployment group load balancer info	The load balancer production listener, optional test listener, and target groups specified are added to the deployment group configuration.

Resource	Default Setting
Traffic rerouting settings	Traffic rerouting – The default setting is Reroute traffic immediately . You can change it on the CodeDeploy console or by updating the <code>TrafficRoutingConfig</code> . For more information, see CreateDeploymentConfig 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.
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 \(p. 674\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy>CreateApplication",
        "codedeploy>CreateDeployment",
        "codedeploy>CreateDeploymentGroup",
        "codedeploy:GetApplication",
        "codedeploy:GetDeployment",
        "codedeploy:ListApplications",
        "codedeploy:ListDeployments",
        "codedeploy:ListDeploymentGroups"
      ]
    }
  ]
}
```

```
        "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 \(p. 704\)](#).

The [Create Service Example \(p. 672\)](#) and [Update Service Example \(p. 673\)](#) 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 \(p. 604\)](#).

`key`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: No

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

`value`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

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

`enableECSManagedTags`

Specifies whether to use Amazon ECS managed tags for the tasks within the service. For more information, see [Tagging your resources for billing \(p. 606\)](#).

`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.

`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 \(p. 531\)](#).

`placementConstraints`

An array of placement constraint objects to use for tasks in your service. You can specify a maximum of 10 constraints per task (this limit includes constraints in the task definition and those specified at run time). If you are using the Fargate launch type, task placement constraints aren't supported.

placementStrategy

The placement strategy objects to use for tasks in your service. You can specify a maximum of four strategy rules per service.

The following is an example service definition for creating a service using an external deployment controller.

```
{  
    "cluster": "",  
    "serviceName": "",  
    "desiredCount": 0,  
    "role": "",  
    "deploymentConfiguration": {  
        "maximumPercent": 0,  
        "minimumHealthyPercent": 0  
    },  
    "placementConstraints": [  
        {  
            "type": "distinctInstance",  
            "expression": ""  
        }  
    ],  
    "placementStrategy": [  
        {  
            "type": "binpack",  
            "field": ""  
        }  
    ],  
    "healthCheckGracePeriodSeconds": 0,  
    "schedulingStrategy": "REPLICA",  
    "deploymentController": {  
        "type": "EXTERNAL"  
    },  
    "tags": [  
        {  
            "key": "",  
            "value": ""  
        }  
    ],  
    "enableECSManagedTags": true,  
    "propagateTags": "TASK_DEFINITION"  
}
```

2. Create an initial task set. The task set contains the following details about your service:

taskDefinition

The task definition for the tasks in the task set to use.

launchType

Type: String

Valid values: EC2 | FARGATE | 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 \(p. 247\)](#).

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 \(p. 169\)](#).

Note

Platform versions are not specified for tasks using the EC2 launch type.

loadBalancers

A load balancer object representing the load balancer to use with your service. When using an external deployment controller, only Application Load Balancers and Network Load Balancers are supported. If you're using an Application Load Balancer, only one Application Load Balancer target group is allowed per task set.

The following snippet shows an example `loadBalancer` object to use.

```
"loadBalancers": [
    {
        "targetGroupArn": "",
        "containerName": "",
        "containerPort": 0
    }
]
```

Note

When specifying a `loadBalancer` object, you must specify a `targetGroupArn` and omit the `loadBalancerName` parameters.

networkConfiguration

The network configuration for the service. This parameter is required for task definitions that use the `awsvpc` network mode to receive their own elastic network interface, and it's not supported for other network modes. For more information, see [Amazon ECS task networking \(p. 278\)](#).

serviceRegistries

The details of the service discovery registries to assign to this service. For more information, see [Service Discovery \(p. 599\)](#).

scale

A floating-point percentage of the desired number of tasks to place and keep running in the task set. The value is specified as a percent total of a service's `desiredCount`. Accepted values are numbers between 0 and 100.

The following is a JSON example for creating a task set for an external deployment controller.

```
{
```

```

    "service": "",
    "cluster": "",
    "externalId": "",
    "taskDefinition": "",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                ""
            ],
            "securityGroups": [
                ""
            ],
            "assignPublicIp": "DISABLED"
        }
    },
    "loadBalancers": [
        {
            "targetGroupArn": "",
            "containerName": "",
            "containerPort": 0
        }
    ],
    "serviceRegistries": [
        {
            "registryArn": "",
            "port": 0,
            "containerName": "",
            "containerPort": 0
        }
    ],
    "launchType": "EC2",
    "capacityProviderStrategy": [
        {
            "capacityProvider": "",
            "weight": 0,
            "base": 0
        }
    ],
    "platformVersion": "",
    "scale": {
        "value": null,
        "unit": "PERCENT"
    },
    "clientToken": ""
}

```

3. When service changes are needed, use the `UpdateService`, `UpdateTaskSet`, or `CreateTaskSet` API action depending on which parameters you're updating. If you created a task set, use the `scale` parameter for each task set in a service to determine how many tasks to keep running in the service. For example, if you have a service that contains `tasksetA` and you create a `tasksetB`, you might test the validity of `tasksetB` before wanting to transition production traffic to it. You could set the `scale` for both task sets to 100, and when you were ready to transition all production traffic to `tasksetB`, you could update the `scale` for `tasksetA` to 0 to scale it down.

Service load balancing

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

Amazon ECS services hosted on Amazon EC2 instances support the Application Load Balancer, Network Load Balancer, and Classic Load Balancer load balancer types. Amazon ECS services hosted on AWS

Fargate support Application Load Balancer and Network Load Balancer only. 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. Classic Load Balancers are used to route TCP traffic. For more information, see [Load balancer types \(p. 576\)](#).

Application Load Balancers offer several features that make them attractive for use with Amazon ECS services:

- Each service can serve traffic from multiple load balancers and expose multiple load balanced ports by specifying multiple target groups.
- They are supported by tasks hosted on both Fargate and EC2 instances.
- Application Load Balancers allow containers to use dynamic host port mapping (so that multiple tasks from the same service are allowed per container instance).
- Application Load Balancers support path-based routing and priority rules (so that multiple services can use the same listener port on a single Application Load Balancer).

We recommend that you use Application Load Balancers for your Amazon ECS services so that you can take advantage of these latest features, unless your service requires a feature that is only available with Network Load Balancers or Classic Load Balancers. For more information about Elastic Load Balancing and the differences between the load balancer types, see the [Elastic Load Balancing User Guide](#).

With your load balancer, you pay only for what you use. For more information, see [Elastic Load Balancing pricing](#).

Topics

- [Service load balancing considerations \(p. 575\)](#)
- [Load balancer types \(p. 576\)](#)
- [Creating a load balancer \(p. 579\)](#)
- [Registering multiple target groups with a service \(p. 589\)](#)

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 \(p. 685\)](#).
- For services that use an Application Load Balancer or Network Load Balancer, you cannot attach more than five target groups to a service.
- For services with tasks using the awsvpc network mode, when you create a target group for your service, you must choose `ip` as the target type, not `instance`. This is because tasks that use the awsvpc network mode are associated with an elastic network interface, not an Amazon EC2 instance.
- If your service 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*.

- Your load balancer subnet configuration must include all Availability Zones that your container instances reside in.
- After you create a service, the target group ARN or load balancer name, container name, and container port specified in the service definition are immutable. You cannot add, remove, or change the load balancer configuration of an existing service. If you update the task definition for the service, the container name and container port that were specified when the service was created must remain in the task definition.
- If a service's task fails the load balancer health check criteria, the task is stopped and restarted. This process continues until your service reaches the number of desired running tasks.
- 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).
- If you are experiencing problems with your load balancer-enabled services, see [Troubleshooting service load balancers](#) (p. 826).

Classic Load Balancer considerations

The following considerations are specific to Amazon ECS services using Classic Load Balancers:

- Services with tasks that use the `awsvpc` network mode, such as those with the Fargate launch type, do not support Classic Load Balancers.
- All of the containers that are launched in a single task definition are always placed on the same container instance. For Classic Load Balancers, you may choose to put multiple containers (in the same task definition) behind the same load balancer by defining multiple host ports in the service definition and adding those listener ports to the load balancer. For example, if a task definition consists of OpenSearch Service using port 3030 on the container instance, with Logstash and OpenSearch Dashboards using port 4040 on the container instance, the same load balancer can route traffic to OpenSearch Service and OpenSearch Dashboards through two listeners. For more information, see [Listeners for your Classic Load Balancer](#) in the *User Guide for Classic Load Balancers*.

Important

We do not recommend connecting multiple services to the same Classic Load Balancer. Because entire container instances are registered and deregistered with Classic Load Balancers, and not with host and port combinations, this configuration can cause issues if a task from one service stops. In this scenario, a task from one service stopping can cause the entire container instance to be deregistered from the Classic Load Balancer while another task from a different service on the same container instance is still using it. If you want to connect multiple services to a single load balancer we recommend using an Application Load Balancer.

Load balancer types

Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. Amazon ECS services can use 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.

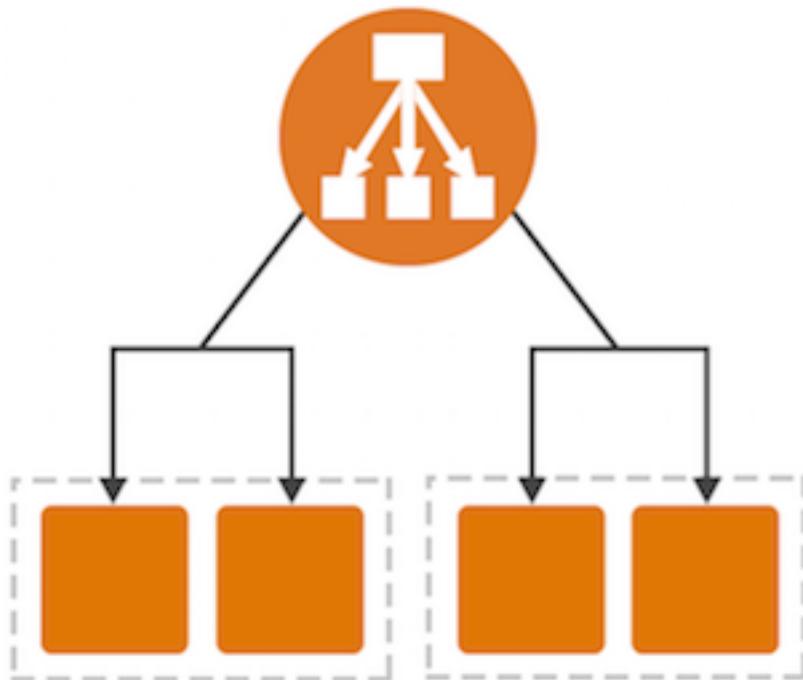
Topics

- [Application Load Balancer](#) (p. 577)

- [Network Load Balancer \(p. 577\)](#)
- [Classic Load Balancer \(p. 578\)](#)
- [Gateway Load Balancers \(p. 579\)](#)

Application Load Balancer

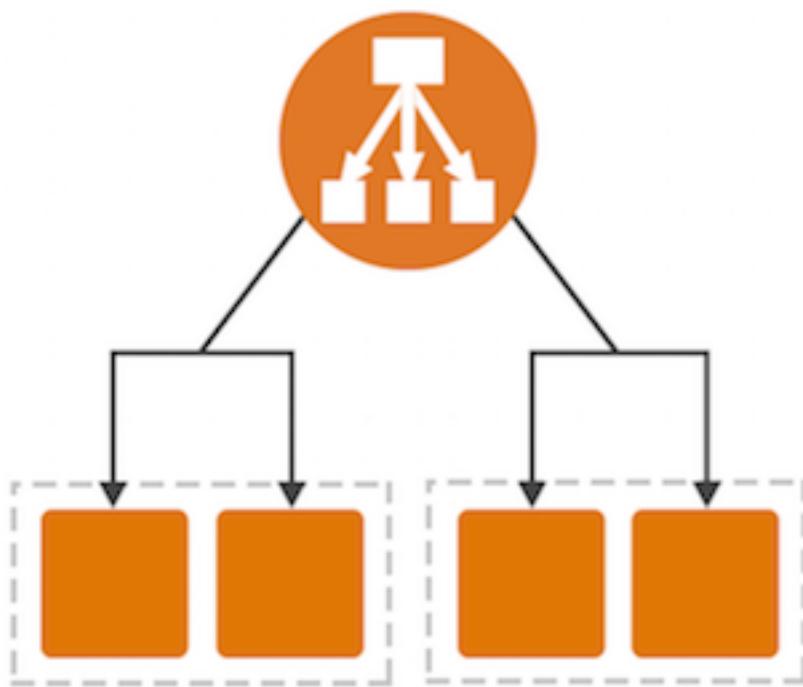
An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports on each container instance in your cluster. Application Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task is launched, the NGINX container is registered with the Application Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see the [User Guide for Application Load Balancers](#).



Network Load Balancer

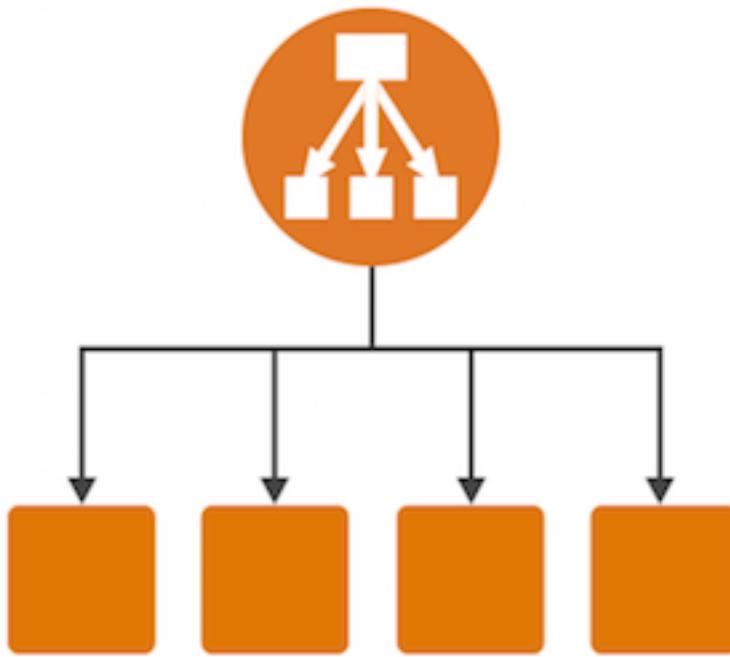
A Network Load Balancer makes routing decisions at the transport layer (TCP/SSL). It can handle millions of requests per second. After the load balancer receives a connection, it selects a target from the target group for the default rule using a flow hash routing algorithm. It attempts to open a TCP connection to the selected target on the port specified in the listener configuration. It forwards the request without modifying the headers. Network Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance (such as 32768 to 61000 on the latest Amazon ECS-optimized AMI). When the task is launched, the NGINX container is registered with the Network Load Balancer as an instance ID and port combination,

and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance. For more information, see the [User Guide for Network Load Balancers](#).



Classic Load Balancer

A Classic Load Balancer makes routing decisions at either the transport layer (TCP/SSL) or the application layer (HTTP/HTTPS). Classic Load Balancers currently require a fixed relationship between the load balancer port and the container instance port. For example, it is possible to map the load balancer port 80 to the container instance port 3030 and the load balancer port 4040 to the container instance port 4040. However, it is not possible to map the load balancer port 80 to port 3030 on one container instance and port 4040 on another container instance. This static mapping requires that your cluster has at least as many container instances as the desired count of a single service that uses a Classic Load Balancer. For more information, see the [User Guide for Classic Load Balancers](#).



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 \(p. 685\)](#).

Topics

- [Creating an Application Load Balancer \(p. 580\)](#)
- [Creating a Network Load Balancer \(p. 584\)](#)
- [Creating a Classic Load Balancer \(p. 585\)](#)

Creating an Application Load Balancer

This section walks you through the process of creating an Application Load Balancer in the AWS Management Console.

Define your load balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and a port for the frontend (client to load balancer) connections, and protocol and a port for the backend (load balancer to backend instance) connections. In this example, you configure a listener that accepts HTTP requests on port 80 and sends them to the containers in your tasks on port 80 using HTTP.

To define your load balancer

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select a Region for your load balancer. Be sure to select the same Region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.
4. Choose **Create Load Balancer**.
5. On the **Select load balancer type** page, choose **Application Load Balancer** and then choose **Continue**.
6. Complete the **Configure Load Balancer** page as follows:
 - a. For **Name**, type a name for your load balancer.
 - b. For **Scheme**, an internet-facing load balancer routes requests from clients over the internet to targets. An internal load balancer routes requests to targets using private IP addresses.
 - c. For **IP address type**, choose **ipv4** to support IPv4 addresses only or **dualstack** to support both IPv4 and IPv6 addresses.

- d. For **Listeners**, the default is a listener that accepts HTTP traffic on port 80. You can keep the default listener settings, modify the protocol or port of the listener, or choose **Add** to add another listener.

Note

If you plan on routing traffic to more than one target group, see [ListenerRules](#) for details on how to add host or path-based rules.

- e. For **VPC**, select the same VPC that you used for the container instances on which you intend to run your service.
- f. For **Availability Zones**, select the check box for the Availability Zones to 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.
- g. Choose **Next: Configure Security Settings**.

Configure security settings

If you created a secure listener in the previous step, complete the **Configure Security Settings** page as follows; otherwise, choose **Next: Configure Security Groups**.

To configure security settings

1. If you have a certificate from AWS Certificate Manager, choose **Choose an existing certificate from AWS Certificate Manager (ACM)**, and then choose the certificate from **Certificate name**.
2. If you have already uploaded a certificate using IAM, choose **Choose an existing certificate from AWS Identity and Access Management (IAM)**, and then choose your certificate from **Certificate name**.
3. If you have a certificate ready to upload, choose **Upload a new SSL Certificate to AWS Identity and Access Management (IAM)**. For **Certificate name**, type a name for the certificate. For **Private Key**, copy and paste the contents of the private key file (PEM-encoded). In **Public Key Certificate**, copy and paste the contents of the public key certificate file (PEM-encoded). In **Certificate Chain**, copy and paste the contents of the certificate chain file (PEM-encoded), unless you are using a self-signed certificate and it's not important that browsers implicitly accept the certificate.
4. For **Select policy**, choose a predefined security policy. For details on the security policies, see [Security Policies](#) in the *User Guide for Application Load Balancers*.
5. Choose **Next: Configure Security Groups**.

Configure security groups

You must assign a security group to your load balancer that allows inbound traffic to the ports that you specified for your listeners. Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

To assign a security group to your load balancer

1. On the **Assign Security Groups** page, choose **Create a new security group**.
2. Enter a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your listener to use.

Note

Later in this topic, you create a security group rule for your container instances that allows traffic on all ports coming from the security group created here, so that the Application

Load Balancer can route traffic to dynamically assigned host ports on your container instances.

Assign a security group: Create a new security group
 Select an existing security group

Security group name: alb-example

Description: Port 80 for HTTP ECS service

Type	Protocol	Port Range	So
HTTP	TCP	80	A

Add Rule

3. Choose **Next: Configure Routing** to go to the next page in the wizard.

Configure routing

In this section, you create a target group for your load balancer and the health check criteria for targets that are registered within that group.

To create a target group and configure health checks

1. For **Target group**, keep the default, **New target group**.
2. For **Name**, type a name for the new target group.
3. Set **Protocol** and **Port** as needed.
4. For **Target type**, choose whether to register your targets with an instance ID or an IP address.

Important

If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

5. For **Health checks**, keep the default health check settings.
6. Choose **Next: Register Targets**.

Register targets

Your load balancer distributes traffic between the targets that are registered to its target groups. When you associate a target group to an Amazon ECS service, Amazon ECS automatically registers and

deregisters containers with your target group. Because Amazon ECS handles target registration, you do not add targets to your target group at this time.

To skip target registration

1. In the **Registered instances** section, ensure that no instances are selected for registration.
2. Choose **Next: Review** to go to the next page in the wizard.

Review and create

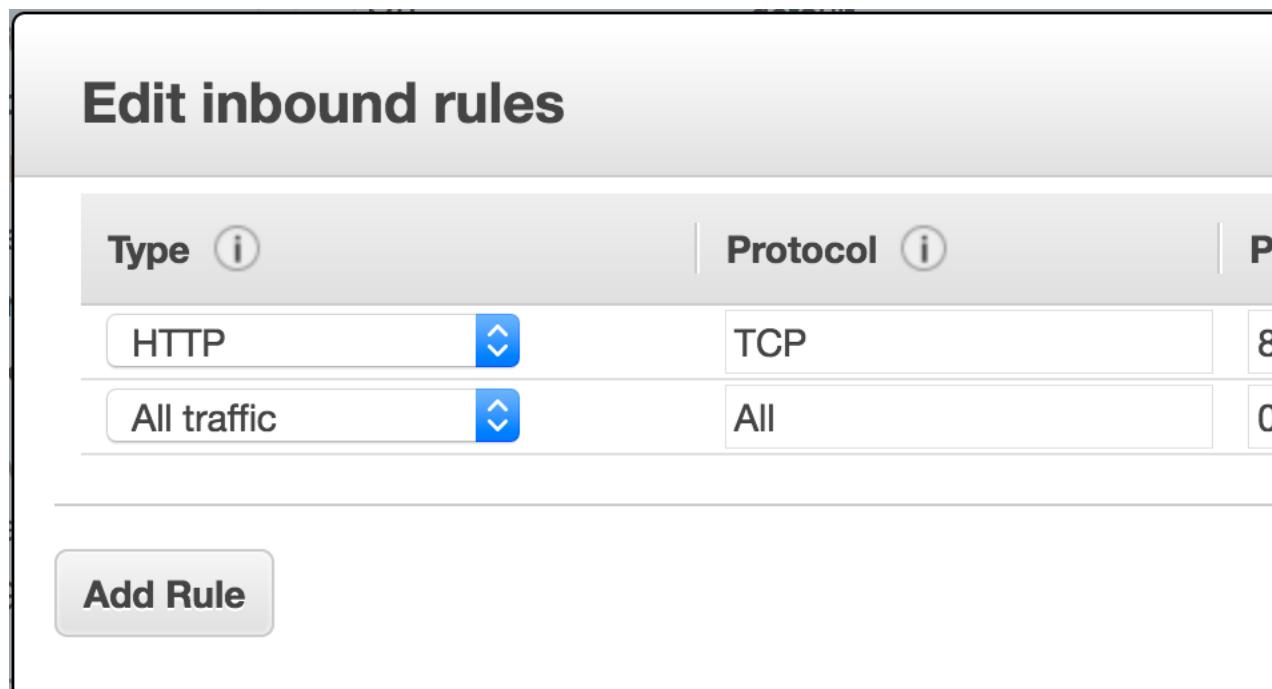
Review your load balancer and target group configuration and choose **Create** to create your load balancer.

Create a security group rule for your container instances

After your Application Load Balancer has been created, you must add an inbound rule to your container instance security group that allows traffic from your load balancer to reach the containers.

To allow inbound traffic from your load balancer to your container instances

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the left navigation, choose **Security Groups**.
3. Choose the security group that your container instances use. If you created your container instances by using the Amazon ECS first run wizard, this security group may have the description, **ECS Allowed Ports**.
4. Choose the **Inbound** tab, and then choose **Edit**.
5. For **Type**, choose **All traffic**.
6. For **Source**, choose **Custom**, and then type the name of your Application Load Balancer security group that you created in [Configure security groups \(p. 581\)](#). 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 \(p. 546\)](#).

Creating a Network Load Balancer

This section walks you through the process of creating a Network Load Balancer in the AWS Management Console.

Define your load balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and port for the frontend (client to load balancer) connections, and a protocol and port for the backend (load balancer to backend instance) connections. In this example, you configure an Internet-facing load balancer in the selected network with a listener that receives TCP traffic on port 80.

To define your load balancer

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.
4. Choose **Create Load Balancer**.
5. On the **Select load balancer type** page, choose **Create** under **Network Load Balancer**.
6. Complete the **Configure Load Balancer** page as follows:
 - a. For **Name**, type a name for your load balancer.
 - b. For **Scheme**, choose either **internet-facing** or **internal**. An internet-facing load balancer routes requests from clients over the internet to targets. An internal load balancer routes requests to targets using private IP addresses.
 - c. For **Listeners**, the default is a listener that accepts TCP traffic on port 80. You can keep the default listener settings, modify the protocol or port of the listener, or choose **Add listener** to add another listener.
 - d. For **Availability Zones**, select the VPC that you used for your Amazon EC2 instances. For each Availability Zone that you used to launch your Amazon EC2 instances, select an Availability Zone and then select the public subnet for that Availability Zone. To associate an Elastic IP address with the subnet, select it from **Elastic IP**.
 - e. Choose **Next: Configure Routing**.

Configure routing

You register targets, such as Amazon EC2 instances, with a target group. The target group that you configure in this step is used as the target group in the listener rule, which forwards requests to the target group. For more information, see [Target Groups for Your Network Load Balancers](#) in the *User Guide for Network Load Balancers*.

To configure your target group

1. For **Target group**, keep the default, **New target group**.
2. For **Name**, type a name for the target group.
3. Set **Protocol** and **Port** as needed.
4. For **Target type**, choose whether to register your targets with an instance ID or an IP address.

Important

If your service's task definition uses the `awsvpc` network mode (which is required for the Fargate launch type), you must choose `ip` as the target type, not `instance`. This is because tasks that use the `awsvpc` network mode are associated with an elastic network interface, not an Amazon EC2 instance.

You cannot register instances by instance ID if they have the following instance types: C1, CC1, CC2, CG1, CG2, CR1, G1, G2, HI1, HS1, M1, M2, M3, and T1. You can register instances of these types by IP address.

5. For **Health checks**, keep the default health check settings.
6. Choose **Next: Register Targets**.

Register targets with the target group

Your load balancer distributes traffic between the targets that are registered to its target groups. When you associate a target group to an Amazon ECS service, Amazon ECS automatically registers and deregisters containers with your target group. Because Amazon ECS handles target registration, you do not add targets to your target group at this time.

To skip target registration

1. In the **Registered instances** section, ensure that no instances are selected for registration.
2. Choose **Next: Review** to go to the next page in the wizard.

Review and create

Review your load balancer and target group configuration and choose **Create** to create your load balancer.

Create an Amazon ECS service

After your load balancer and target group are created, you can specify the target group in a service definition when you create a service. When each task for your service is started, the container and port combination specified in the service definition is registered with your target group and traffic is routed from the load balancer to that container. For more information, see [Creating an Amazon ECS service \(p. 546\)](#).

Creating a Classic Load Balancer

This section walks you through the process of creating a Classic Load Balancer in the AWS Management Console.

You can create your Classic Load Balancer for use with EC2-Classic or a VPC. Some of the tasks described in these procedures apply only to load balancers in a VPC.

Define your load balancer

First, provide some basic configuration information for your load balancer, such as a name, a network, and a listener.

A *listener* is a process that checks for connection requests. It is configured with a protocol and port for the frontend (client to load balancer) connections and a protocol, and a protocol and port for the backend (load balancer to backend instance) connections. In this example, you configure a listener that accepts HTTP requests on port 80 and sends them to the backend instances on port 80 using HTTP.

To define your load balancer

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select a region for your load balancer. Be sure to select the same region that you selected for your Amazon ECS container instances.
3. In the navigation pane, under **LOAD BALANCING**, choose **Load Balancers**.
4. Choose **Create Load Balancer**.
5. On the **Select load balancer type** page, choose **Classic Load Balancer**.
6. For **Load Balancer name**, enter a unique name for your load balancer.

The load balancer name you choose must be unique within your set of load balancers, must have a maximum of 32 characters, and must only contain alphanumeric characters or hyphens.

7. For **Create LB inside**, select the same network that your container instances are located in: EC2-Classic or a specific VPC.
8. The default values configure an HTTP load balancer that forwards traffic from port 80 at the load balancer to port 80 of your container instances, but you can modify these values for your application. For more information, see [Listeners for Your Classic Load Balancer](#) in the *User Guide for Classic Load Balancers*.
9. [EC2-VPC] To improve the availability of your load balancer, select at least two subnets in different Availability Zones. Your load balancer subnet configuration must include all Availability Zones that your container instances reside in. In the **Select Subnets** section, under **Available Subnets**, select the subnets. The subnets that you select are moved under **Selected Subnets**.

Note

If you selected EC2-Classic as your network, or you have a default VPC but did not choose **Enable advanced VPC configuration**, you do not see **Select Subnets**.

Available Subnets

Actions	Availability Zone	Subnet ID	Subnet CIDR	Name
	us-west-2c	subnet-cb663da2	10.0.1.0/24	
	us-west-2c	subnet-c9663da0	10.0.0.0/24	

Selected Subnets

Actions	Availability Zone	Subnet ID	Subnet CIDR	Name
	us-west-2a	subnet-e4f33493	10.0.2.0/24	
	us-west-2b	subnet-5264e837	10.0.3.0/24	

10. Choose **Next: Assign Security Groups** to go to the next page in the wizard.

Assign a security group to your load balancer in a VPC

If you created your load balancer in a VPC, you must assign it a security group that allows inbound traffic to the ports that you specified for your load balancer and the health checks for your load balancer.

Amazon ECS does not automatically update the security groups associated with Elastic Load Balancing load balancers or Amazon ECS container instances.

Note

If you selected EC2-Classic as your network, you do not see this page in the wizard and you can go to the next step. Elastic Load Balancing provides a security group that is assigned to your load balancer for EC2-Classic automatically.

To assign a security group to your load balancer

1. On the **Assign Security Groups** page, choose **Create a new security group**.
2. Enter a name and description for your security group, or leave the default name and description. This new security group contains a rule that allows traffic to the port that you configured your load balancer to use. If you specified a different port for the health checks, you must choose **Add Rule** to add a rule that allows inbound traffic to that port as well.

Note

Also assign this security group to container instances in your service, or another security group with the same rules.

Assign Security Groups

Assign a security group:

Create a new security group
 Select an existing security group

Security group name: my-lb-group

Description: created for getting started tutorial

Type	Protocol	Port Range
Custom TCP Rule	TCP	80

Add Rule

3. Choose **Next: Configure Security Settings** to go to the next page in the wizard.

Configure security settings

For this tutorial, you can choose **Next: Configure Health Check** to continue to the next step. For more information about creating an HTTPS load balancer and using additional security features, see [HTTPS Load Balancers](#) in the *User Guide for Classic Load Balancers*.

Configure health checks for your Amazon EC2 instances

Elastic Load Balancing automatically checks the health of the tasks in your service. If Elastic Load Balancing finds an unhealthy task, it stops sending traffic to the Amazon EC2 instance hosting that task and reroutes the traffic to a healthy instance.

Note

The following procedure configures an HTTP (port 80) load balancer, but you can modify these values for your application.

To configure a health check for your instances

1. On the **Configure Health Check** page, do the following:

- a. Leave **Ping Protocol** set to its default value of **HTTP**.
- b. Leave **Ping Port** set to its default value of **80**.
- c. For **Ping Path**, replace the default value with a single forward slash (""). This tells Elastic Load Balancing to send health check queries to the default home page for your web server, such as `index.html` or `default.html`.
- d. Leave the other fields at their default values.

Configure Health Check

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check settings to meet your specific needs.

Ping Protocol	<input style="border: 1px solid #ccc; padding: 2px 5px; width: 100px; height: 20px; border-radius: 5px; font-size: 10px; font-weight: bold; text-decoration: none; color: inherit; background-color: inherit;" type="button" value="HTTP"/>
Ping Port	<input style="width: 100px; height: 20px; border: 1px solid #ccc; padding: 2px; border-radius: 5px; font-size: 10px; font-weight: bold; text-decoration: none; color: inherit; background-color: inherit;" type="text" value="80"/>
Ping Path	<input style="width: 100px; height: 20px; border: 1px solid #ccc; padding: 2px; border-radius: 5px; font-size: 10px; font-weight: bold; text-decoration: none; color: inherit; background-color: inherit;" type="text" value="/"/>

2. Choose **Next: Add EC2 Instances** to go to the next page in the wizard.

Load balancer instance registration

Your load balancer distributes traffic between the instances that are registered to it. When you assign your load balancer to an Amazon ECS service, Amazon ECS automatically registers and deregisters container instances when tasks from your service are running on them. Because Amazon ECS handles container instance registration, you do not add container instances to your load balancer at this time.

To skip instance registration and tag the load balancer

1. On the **Add EC2 Instances** page, for **Add Instances to Load Balancer**, ensure that no instances are selected for registration.
2. Leave the other fields at their default values.
3. Choose **Next: Add Tags** to go to the next page in the wizard.

Tag your load balancer

You can tag your load balancer, or continue to the next step. You can tag your load balancer later on. For more information, see [Tag Your Classic Load Balancer](#) in the *User Guide for Classic Load Balancers*.

To add tags to your load balancer

1. On the **Add Tags** page, specify a key and a value for the tag.
2. To add another tag, choose **Create Tag** and specify a key and a value for the tag.
3. After you are finished adding tags, choose **Review and Create**.

Create and verify your load balancer

Before you create the load balancer, review the settings that you selected. After creating the load balancer, you can create a service that uses it to verify that it's sending traffic to your container instances.

To finish creating your load balancer

1. On the **Review** page, check your settings. To change the initial settings, choose the corresponding edit link.
2. Choose **Create** to create your load balancer.
3. After you are notified that your load balancer was created, choose **Close**.

Create an Amazon ECS service

After your load balancer is created, you can specify it in a service definition when you create a service. For more information, see [Creating an Amazon ECS service \(p. 546\)](#).

Registering multiple target groups with a service

Your Amazon ECS service can serve traffic from multiple load balancers and expose multiple load balanced ports when you specify multiple target groups in a service definition.

To create a service specifying multiple target groups, you must create the service using the Amazon ECS API, SDK, AWS CLI, or an AWS CloudFormation template. After the service is created, you can view the service and the target groups registered to it with the AWS Management Console. It is not possible to update the load balancing configuration of an existing service.

Multiple target groups can be specified in a service definition using the following format. For the full syntax of a service definition, see [Service definition template \(p. 545\)](#).

```
"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 \(p. 685\)](#).

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 \(p. 545\)](#).

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 \(p. 625\)](#). 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 \(p. 593\)](#)—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 \(p. 597\)](#)—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 \(p. 672\)](#) and [Update Service Example \(p. 673\)](#) 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*.

Note that if you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, you might need to add the `ecs:StartTelemetrySession` permission. For more information, see [Enabling CloudWatch metrics \(p. 620\)](#).

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 \(p. 591\)](#).
- You can have multiple target tracking scaling policies for an Amazon ECS service, provided that each of them uses a different metric. The intention of Service Auto Scaling is to always prioritize availability, so its behavior differs depending on whether the target tracking policies are ready for scale out or scale in. It will scale out the service if any of the target tracking policies are ready for scale out, but will scale in only if all of the target tracking policies (with the scale-in portion enabled) are ready to scale in.
- Do not edit or delete the CloudWatch alarms that Service Auto Scaling manages for a target tracking scaling policy. Service Auto Scaling deletes the alarms automatically when you delete the scaling policy.

Tutorial: Service auto scaling with target tracking

The following procedures help you to create an Amazon ECS cluster and a service that uses target tracking to scale out (and in) automatically based on demand.

In this tutorial, you use the Amazon ECS first-run wizard to create a cluster and a service that runs behind an Elastic Load Balancing load balancer. Then you configure a target tracking scaling policy that scales your service automatically based on the current application load as measured by the service's CPU utilization (from the **ECS, ClusterName, ServiceName** category in CloudWatch).

When the average CPU utilization of your service rises above 75% (meaning that more than 75% of the CPU that is reserved for the service is being used), a scale-out alarm triggers Service Auto Scaling to add another task to your service to help out with the increased load. Conversely, when the average CPU utilization of your service drops below the target utilization for a sustained period of time, a scale-in alarm triggers a decrease in the service's desired count to free up those cluster resources for other tasks and services.

Prerequisites

This tutorial assumes that you are using administrator credentials, and that you have an Amazon EC2 key pair in the current region. If you do not have these resources, or your are not sure, you can create them by following the steps in [Setting up with Amazon ECS \(p. 7\)](#).

Step 1: Create a cluster and a service

Start by creating a cluster and service using the Amazon ECS first-run wizard. The first-run wizard takes care of creating the necessary IAM roles for this tutorial, an Auto Scaling group for your container instances, and a service that runs behind a load balancer. The wizard also makes the clean-up process much easier, because you can delete the entire AWS CloudFormation stack in one step.

For this tutorial, you create a cluster called `service-autoscaling` and a service called `sample-webapp`.

To create your cluster and service

1. Open the Amazon ECS console first run wizard at <https://console.aws.amazon.com/ecs/home#/firstRun>.
2. From the navigation bar, choose the **US East (N. Virginia)** region.
3. On **Step 1: Container and Task**, for **Container definition**, select `sample-app`.
4. For **Task definition**, leave all of the default options and choose **Next**.
5. On **Step 2: Service**, for **Load balancer type**, choose **Application Load Balancer**, **Next**.

Important

Application Load Balancers do incur costs while they exist in your AWS resources. For more information, see [Elastic Load Balancing Pricing](#).

6. On **Step 3: Cluster**, for **Cluster name**, enter `service-autoscaling` and choose **Next**.
7. Review your choices and then choose **Create**.

You are directed to a **Launch Status** page that shows the status of your launch and describes each step of the process (this can take a few minutes to complete while your cluster resources are created and populated).

8. When your cluster and service are created, choose **View service**.

Step 2: Configure service auto scaling

Now that you have launched a cluster and created a service in that cluster that is running behind a load balancer, you can 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	M
Load Balancing					
Target Group Name	Container Name	Con			
EC2Co-Defau-13FL25TVMRZRO	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 \(p. 645\)](#) and [Updating cluster settings \(p. 192\)](#).

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 \(p. 593\)](#).

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, which is what processing capacity is supposed to be. This prevents excessive (runaway) scaling that could not be satisfied, for example, if there are not enough container instance resources to place the additional tasks. If the container instance capacity is available later, the pending scaling activity may succeed, and then further scaling activities can continue after the cooldown period.
- If you want your task count to scale to zero when there is 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 \(p. 591\)](#).

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 \(p. 546\)](#) and [Updating a service \(p. 559\)](#). For more information about step scaling and a walkthrough, see [Automatic Scaling with Amazon ECS](#) in the *AWS Compute Blog*. For a target tracking walkthrough, see [Target tracking scaling policies \(p. 593\)](#).

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

Your Amazon ECS service can optionally be configured to use Amazon ECS Service Discovery. Service discovery uses AWS Cloud Map API actions to manage HTTP and DNS namespaces for your Amazon ECS services. For more information, see [What Is AWS Cloud Map?](#) in the *AWS Cloud Map Developer Guide*.

Service discovery is available in the following AWS Regions:

Region Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Tokyo)	ap-northeast-1

Region Name	Region
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
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`.
- **Service discovery service:** Exists within the service discovery namespace and consists of the service name and DNS configuration for the namespace. It provides the following core component:
 - **Service registry:** Allows you to look up a service via DNS or AWS Cloud Map API actions and get back one or more available endpoints that can be used to connect to the service.
- **Service discovery instance:** Exists within the service discovery service and consists of the attributes associated with each Amazon ECS service in the service directory.
 - **Instance attributes:** The following metadata is added as custom attributes for each Amazon ECS service that is configured to use service discovery:
 - **AWS_INSTANCE_IPV4** – For an A record, the IPv4 address that Route 53 returns in response to DNS queries and AWS Cloud Map returns when discovering instance details, for example, `192.0.2.44`.
 - **AWS_INSTANCE_PORT** – The port value associated with the service discovery service.
 - **AVAILABILITY_ZONE** – The Availability Zone into which the task was launched. For tasks using the EC2 launch type, this is the Availability Zone in which the container instance exists. For tasks using the Fargate launch type, this is the Availability Zone in which the elastic network interface exists.
 - **REGION** – The Region in which the task exists.

- **ECS_SERVICE_NAME** – The name of the Amazon ECS service to which the task belongs.
- **ECS_CLUSTER_NAME** – The name of the Amazon ECS cluster to which the task belongs.
- **EC2_INSTANCE_ID** – The ID of the container instance the task was placed on. This custom attribute is not added if the task is using the Fargate launch type.
- **ECS_TASK_DEFINITION_FAMILY** – The task definition family that the task is using.
- **ECS_TASK_SET_EXTERNAL_ID** – If a task set is created for an external deployment and is associated with a service discovery registry, then the **ECS_TASK_SET_EXTERNAL_ID** attribute will contain the external ID of the task set.
- **Amazon ECS health checks:** Amazon ECS performs periodic container-level health checks. If an endpoint does not pass the health check, it is removed from DNS routing and marked as unhealthy.

Service discovery considerations

The following should be considered when using service discovery:

- Service discovery is supported for tasks on Fargate that use platform version 1.1.0 or later. For more information, see [AWS Fargate platform versions \(p. 169\)](#).
- Services configured to use service discovery have a limit of 1,000 tasks per service. This is due to a Route 53 service quota.
- The Create Service workflow in the Amazon ECS console only supports registering services into private DNS namespaces. When an AWS Cloud Map private DNS namespace is created, a Route 53 private hosted zone will be created automatically.
- The VPC DNS attributes must be configured for successful DNS resolution. For information about how to configure the attributes, see [DNS support in your VPC](#) in the *Amazon VPC User Guide*.
- The DNS records created for a service discovery service always register with the private IP address for the task, rather than the public IP address, even when public namespaces are used.
- Service discovery requires that tasks specify either the `awsvpc`, `bridge`, or `host` network mode (`none` is not supported).
- If the task definition your service task specifies uses the `awsvpc` network mode, you can create any combination of A or SRV records for each service task. If you use SRV records, a port is required.
- If the task definition that your service task specifies uses the `bridge` or `host` network mode, an SRV record is the only supported DNS record type. Create an SRV record for each service task. The SRV record must specify a container name and container port combination from the task definition.
- DNS records for a service discovery service can be queried within your VPC. They use the following format: <service discovery service name>. <service discovery namespace>.
- When doing a DNS query on the service name, A records return a set of IP addresses that correspond to your tasks. SRV records return a set of IP addresses and ports per task.
- If you have eight or fewer healthy records, Route 53 responds to all DNS queries with all of the healthy records.
- When all records are unhealthy, Route 53 responds to DNS queries with up to eight unhealthy records.
- You can configure service discovery for an ECS service that is behind a load balancer, but service discovery traffic is always routed to the task and not the load balancer.
- Service discovery does not support the use of Classic Load Balancers.
- It is recommended to use container-level health checks managed by Amazon ECS for your service discovery service.
 - **HealthCheckCustomConfig**—Amazon ECS manages health checks on your behalf. Amazon ECS uses information from container and health checks, as well as your task state, to update the health with AWS Cloud Map. This is specified using the `--health-check-custom-config` parameter when creating your service discovery service. For more information, see [HealthCheckCustomConfig](#) in the *AWS Cloud Map API Reference*.

- If you are using the Amazon ECS console, the workflow creates one service discovery service per ECS service. It maps all of the task IP addresses as A records, or task IP addresses and port as SRV records.
- Service discovery can only be configured when first creating a service. Updating existing services to configure service discovery for the first time or change the current configuration is not supported.
- The AWS Cloud Map resources created when service discovery is used must be cleaned up manually.

Amazon ECS console experience

The service creation workflow in the Amazon ECS console supports service discovery. Service discovery can only be configured when first creating a service. Updating existing services to configure service discovery for the first time or change the current configuration is not supported.

To create a new Amazon ECS service that uses service discovery, see [Creating an Amazon ECS service \(p. 546\)](#).

Service discovery pricing

Customers using Amazon ECS service discovery are charged for Route 53 resources and AWS Cloud Map discovery API operations. This involves costs for creating the Route 53 hosted zones and queries to the service registry. For more information, see [AWS Cloud Map Pricing](#) in the *AWS Cloud Map Developer Guide*.

Amazon ECS performs container level health checks and exposes them to AWS Cloud Map custom health check API operations. This is currently made available to customers at no extra cost. If you configure additional network health checks for publicly exposed tasks, you are charged for those health checks.

Service throttle logic

The Amazon ECS service scheduler includes logic that throttles how often service tasks are launched if they repeatedly fail to start.

If tasks for an ECS service repeatedly fail to enter the `RUNNING` state (progressing directly from `PENDING` to `STOPPED`), then the time between subsequent restart attempts is incrementally increased up to a maximum of 15 minutes. This maximum period is subject to change in the future and should not be considered permanent. This behavior reduces the effect that unstartable tasks have on your Amazon ECS cluster resources or Fargate infrastructure costs. If your service triggers the throttle logic, you receive the following [service event message \(p. 823\)](#):

```
(service service-name) is unable to consistently start tasks successfully.
```

Amazon ECS does not ever stop a failing service from retrying, nor does it attempt to modify it in any way other than increasing the time between restarts. The service throttle logic does not provide any user-tunable parameters.

If you update your service to use a new task definition, your service returns to a normal, non-throttled state immediately. For more information, see [Updating a service \(p. 559\)](#).

The following are some common causes that trigger this logic:

- A lack of resources with which to host your task, such as ports, memory, or CPU units in your cluster. In this case, you also see the [insufficient resource service event message \(p. 821\)](#).
- The Amazon ECS container agent is unable to pull your task Docker image. This could be due to a bad container image name, image, or tag, or a lack of private registry authentication or permissions. In this case, you also see `CannotPullContainerError` in your [stopped task errors \(p. 815\)](#).

- Insufficient disk space on your container instance to create the container. In this case, you also see [CannotCreateContainerError](#) in your [stopped task errors \(p. 815\)](#). For more information, see [CannotCreateContainerError: API error \(500\): devmapper \(p. 825\)](#).

Important

Tasks that are stopped after they reach the RUNNING state do not trigger the throttle logic or the associated service event message. For example, if failed Elastic Load Balancing health checks for a service cause a task to be flagged as unhealthy, and Amazon ECS deregisters it and kills the task, this does not trigger the throttle. Even if a task's container command immediately exits with a non-zero exit code, the task has already moved to the RUNNING state. Tasks that fail immediately due to command errors do not trigger the throttle or the service event message.

Resources and tags

Amazon ECS resources, including task definitions, clusters, tasks, services, and container instances, are assigned an Amazon Resource Name (ARN) and a unique resource identifier (ID). These resources can be tagged with values that you define, to help you organize and identify them.

The following topics provide an overview on these resources and tags and show how you can use them.

Contents

- [Tagging your Amazon ECS resources \(p. 604\)](#)
- [Amazon ECS service quotas \(p. 611\)](#)
- [Supported Regions for Amazon ECS on AWS Fargate \(p. 614\)](#)
- [Amazon ECS usage reports \(p. 617\)](#)

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 \(p. 325\)](#).

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.

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
Amazon ECS container instances	Yes	Yes, from the Amazon EC2 instance. For more information, see Adding tags to an Amazon EC2 container instance (p. 608) .	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 Adding and deleting tags on an individual resource (p. 607) .

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 \(p. 617\)](#).

To see the cost of your combined resources, you can organize your billing information based on resources that have the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information about setting up a cost allocation report with tags, see [The Monthly Cost Allocation Report](#) in the *AWS Billing and Cost Management User Guide*.

Important

To use this feature, 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 \(p. 325\)](#).

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 \(p. 607\)](#)
- [Adding and deleting tags on an individual resource \(p. 607\)](#)
- [Adding tags to an Amazon EC2 container instance \(p. 608\)](#)
- [Adding tags to an external container instance \(p. 609\)](#)

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.	Run a standalone task (p. 510)
Create a service.	Creating an Amazon ECS service (p. 546)
Create a task set.	External deployment (p. 569)
Register a task definition.	Creating a task definition using the new console (p. 196)
Create a cluster.	Creating a cluster (p. 176)
Run one or more container instances.	Launching an Amazon ECS Linux container instance (p. 364)

Adding and deleting tags on an individual resource

Amazon ECS enables you to add or delete tags associated with your clusters, services, tasks, and task definitions directly from the resource's page. For information about tagging your container instances, see [Adding tags to an Amazon EC2 container instance \(p. 608\)](#).

To add a tag to an individual resource

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region to use.

3. In the navigation pane, select a resource type (for example, **Clusters**).
4. Select the resource from the resource list and choose **Tags, Edit**.
5. In the **Edit Tags** dialog box, specify the key and value for each tag, and then choose **Save**.

To delete a tag from an individual resource

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose a resource type (for example, **Clusters**).
4. Select the resource from the resource list and choose **Tags, Edit**.
5. On the **Edit Tags** page, select the **Delete** icon for each tag you want to delete, and choose **Save**.

Adding tags to an Amazon EC2 container instance

You can associate tags with your container instances using one of the following methods:

- Method 1 – When creating the container instance using the Amazon EC2 API, CLI, or console, specify tags by passing user data to the instance using the container agent configuration parameter `ECS_CONTAINER_INSTANCE_TAGS`. This creates tags that are associated with the container instance in Amazon ECS only, they cannot be listed using the Amazon EC2 API. For more information, see [Bootstrapping container instances with Amazon EC2 user data \(p. 368\)](#).

Important

If you launch your container instances using an Amazon EC2 Auto Scaling group, then you should use the `ECS_CONTAINER_INSTANCE_TAGS` agent configuration parameter to add tags. This is due to the way in which tags are added to Amazon EC2 instances that are launched using Auto Scaling groups.

The following is an example of a user data script that would associate tags with your container instance:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
EOF
```

- Method 2 – When creating your container instance using the Amazon EC2 API, CLI, or console, specify tags using the `TagSpecification.N` parameter, and then pass user data to the instance using the container agent configuration parameter `ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM`. Doing so propagates them from Amazon EC2 to Amazon ECS.

The following is an example of a user data script that propagates the tags associated with an Amazon EC2 instance, and registers the instance with a cluster named *MyCluster*:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_CONTAINER_INSTANCE_PROPAGATE_TAGS_FROM=ec2_instance
EOF
```

To provide access to allow container instance tags to propagate from Amazon EC2 to Amazon ECS, manually add the following permissions as an inline policy to the Amazon ECS container instance IAM role. For more information, see [Adding and Removing IAM Policies](#).

- `ec2:DescribeTags`

The following is an example inline policy used to add these permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
    ]
}
```

Adding tags to an external container instance

You can associate tags with your external container instances using one of the following methods.

- Method 1 – Before running the installation script to register your external instance with your cluster, create or edit the Amazon ECS container agent configuration file at `/etc/ecs/ecs.config` and add the `ECS_CONTAINER_INSTANCE_TAGS` container agent configuration parameter. This creates tags that are associated with the external instance.

The following is an example for the syntax that should be used.

```
ECS_CONTAINER_INSTANCE_TAGS={"tag_key": "tag_value"}
```

- Method 2 – After your external instance is registered to your cluster, you can use the AWS Management Console add tags. For more information, see [Adding and deleting tags on an individual resource \(p. 607\)](#).

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.	tag-resource	TagResource
Delete one or more tags.	untag-resource	UntagResource

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing cluster

The following command tags an existing cluster.

```
aws ecs tag-resource --resource-arn resource_ARN --tags key=stack,value=dev
```

Example 2: Untag an existing cluster

The following command deletes a tag from an existing cluster.

```
aws ecs untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Example 3: List tags for a resource

The following command lists the tags associated with an existing resource.

```
aws ecs list-tags-for-resource --resource-arn resource_ARN
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.

Task	AWS CLI	AWS Tools for Windows PowerShell	API Action
Run one or more tasks.	run-task	Start-ECS Task	RunTask
Create a service.	create-service	New-ECS Service	CreateService
Create a task set.	create-task-set	New-ECS Task Set	CreateTaskSet
Register a task definition.	register-task-definition	Register-ECS Task Definition	RegisterTaskDefinition
Create a cluster.	create-cluster	New-ECS Cluster	CreateCluster
Run one or more container instances.	run-instances	New-EC2 Instance	RunInstances

The following examples demonstrate how to apply tags when you create resources.

Example 1: Create a cluster and apply a tag

The following command creates a cluster named `devcluster` and adds a tag with key `team` and value `devs`.

```
aws ecs create-cluster --cluster-name devcluster --tags key=team,value=devs
```

Example 2: Create a service and apply a tag

The following command creates a service named `application` and adds a tag with key `stack` and value `dev`.

```
aws ecs create-service --service-name application --task-definition task-def-app --tags key=stack,value=dev
```

Example 3: Create a service with tags and propagate the tags

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*.

Amazon ECS service quotas

The following are Amazon ECS service quotas.

Most of these service quotas, but not all, are listed under the Amazon Elastic Container Service (Amazon ECS) namespace in the Service Quotas console. To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Service quota	Description	Default quota value	Adjustable
Clusters	The maximum number of clusters in this account in the current Region.	10,000	Yes
Container instances per cluster	The maximum number of container instances per cluster.	2,000	Yes
Services per cluster	The maximum number of services per cluster.	5,000	Yes
Tasks per service	<p>The maximum number of tasks per service (the desired count).</p> <p>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</p>	5,000	Yes

Service quota	Description	Default quota value	Adjustable
	quotas in the Amazon Web Services General Reference.		
ECS Exec sessions	The maximum number of ECS Exec sessions per container.	10	Yes
Tasks launched (count) per run-task	The maximum number of tasks that can be launched per RunTask API action.	10	No
Container instances per start-task	The maximum number of container instances specified in a StartTask API action.	10	No
Revisions per task definition family	The maximum number of revisions per task definition family. Deregistering a task definition revision does not exclude it from being included in this limit.	1,000,000	No
Task definition size limit	The maximum size, in KiB, of a task definition.	64	No
Task definition max containers	The maximum number of containers definitions within a task definition.	10	No
Subnets specified in an <code>awsvpcConfiguration</code>	The maximum number of subnets specified within an <code>awsvpcConfiguration</code> .	16	No
Security groups specified in an <code>awsvpcConfiguration</code>	The maximum number of security groups specified within an <code>awsvpcConfiguration</code> .	5	No
Target groups per service	The maximum number of target groups per service, if using an Application Load Balancer or a Network Load Balancer.	5	No
Classic Load Balancers per service	The maximum number of Classic Load Balancers per service.	1	No

Service quota	Description	Default quota value	Adjustable
Tags per resource	The maximum number of tags per resource. This applies to task definitions, clusters, tasks, and services.	50	No

AWS Fargate service quotas

The following are Amazon ECS on AWS Fargate service quotas.

These service quotas are listed under the AWS Fargate namespace in the Service Quotas console. Quotas for the Fargate On-Demand and Fargate Spot resource count vary. For a new account, the quota can be lower. 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*.

Service quota	Description	Default quota value	Adjustable
Fargate On-Demand resource count	The maximum number of Amazon ECS tasks and Amazon EKS pods running concurrently on Fargate in this account in the current Region.	1,000	Yes
Fargate Spot resource count	The maximum number of Amazon ECS tasks running concurrently on Fargate Spot in this account in the current Region.	1,000	Yes

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 \(p. 614\)](#)
- [Supported Regions for Windows containers on AWS Fargate \(p. 615\)](#)

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
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
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)
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)
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)
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

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.

For tasks using the EC2 launch type, the `lineItem/Operation` column shows `ECSTask-EC2` and the tasks don't have a direct cost associated with them. The metering data shown in the report, such as memory usage, represents the total resources reserved by the task over the billing period indicated. These values can be used to determine the cost of your underlying cluster of Amazon EC2 instances. The cost and usage data for your Amazon EC2 instances are listed separately under the Amazon EC2 service.

You can also use the Amazon ECS-managed tags to identify the service or cluster that each task belongs to. For more information, see [Tagging your resources for billing \(p. 606\)](#).

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 and Cost Management User Guide](#).

Monitoring Amazon ECS

You can monitor your Amazon ECS resources using Amazon CloudWatch, which collects and processes raw data from Amazon ECS into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your clusters or services are performing. Amazon ECS metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the [Amazon CloudWatch User Guide](#).

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon ECS and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. Before you start monitoring Amazon ECS; however, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

The metrics made available depend on the launch type of the tasks and services in your clusters. If you are using the Fargate launch type for your services, then CPU and memory utilization metrics are provided to assist in the monitoring of your services. For the Amazon EC2 launch type, you own and need to monitor the EC2 instances that make up your underlying infrastructure. Additional CPU and memory reservation and utilization metrics are made available at the cluster, service, and task level.

The next step is to establish a baseline for normal Amazon ECS performance in your environment, by measuring performance at various times and under different load conditions. As you monitor Amazon ECS, store historical monitoring data so that you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

To establish a baseline you should, at a minimum, monitor the following items:

- The CPU and memory and reservation utilization metrics for your Amazon ECS clusters
- The CPU and memory utilization metrics for your Amazon ECS services

Topics

- [Monitoring tools \(p. 619\)](#)
- [Amazon ECS CloudWatch metrics \(p. 620\)](#)
- [Amazon ECS events and EventBridge \(p. 632\)](#)
- [Amazon ECS CloudWatch Container Insights \(p. 645\)](#)
- [Container instance health \(p. 647\)](#)
- [Collecting application trace data \(p. 648\)](#)
- [Logging Amazon ECS API calls with AWS CloudTrail \(p. 650\)](#)

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 \(p. 620\)](#).

For clusters with tasks or services using the EC2 launch type, you can use CloudWatch alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation. For more information, see [Tutorial: Scaling container instances with CloudWatch alarms \(p. 628\)](#).

- 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 \(p. 283\)](#).

You can also monitor, store, and access the operating system and Amazon ECS container agent log files from your Amazon ECS container instances. This method for accessing logs can be used for containers using the EC2 launch type. For more information, see [Monitoring your container instances \(p. 426\)](#).

- 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 \(p. 632\)](#) 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 \(p. 650\)](#) in this guide, and [Working with CloudTrail Log Files](#) in the [AWS CloudTrail User Guide](#).

Manual monitoring tools

Another important part of monitoring Amazon ECS involves manually monitoring those items that the CloudWatch alarms don't cover. The CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your container instances and the containers in your tasks.

- CloudWatch home page:
 - Current alarms and status
 - Graphs of alarms and resources
 - Service health status

In addition, you can use CloudWatch to do the following:

- Create [customized dashboards](#) to monitor the services you care about.

- Graph metric data to troubleshoot issues and discover trends.
- Search and browse all your AWS resource metrics.
- Create and edit alarms to be notified of problems.
- AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see [AWS Trusted Advisor](#).

Amazon ECS CloudWatch metrics

You can monitor your Amazon ECS resources using Amazon CloudWatch, which collects and processes raw data from Amazon ECS into readable, near real-time metrics. These statistics are recorded for a period of two weeks so that you can access historical information and gain a better perspective on how your clusters or services are performing. Amazon ECS metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see the [Amazon CloudWatch User Guide](#).

Topics

- [Enabling CloudWatch metrics \(p. 620\)](#)
- [Available metrics and dimensions \(p. 621\)](#)
- [Cluster reservation \(p. 623\)](#)
- [Cluster utilization \(p. 624\)](#)
- [Service utilization \(p. 625\)](#)
- [Service RUNNING task count \(p. 626\)](#)
- [Viewing Amazon ECS metrics \(p. 626\)](#)
- [Tutorial: Scaling container instances with CloudWatch alarms \(p. 628\)](#)

Enabling CloudWatch metrics

Any Amazon ECS service using the Fargate launch type is enabled for CloudWatch CPU and memory utilization metrics automatically, so you don't need to take any manual steps.

For any Amazon ECS task or service using the EC2 launch type, your Amazon ECS container instances require version 1.4.0 or later (Linux) or 1.0.0 or later (Windows) of the container agent to enable CloudWatch metrics. However, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

If you're starting your agent manually (for example, if you're not using the Amazon ECS-optimized AMI for your container instances), see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 452\)](#).

Your Amazon ECS container instances also require the `ecs:StartTelemetrySession` permission on the IAM role that you launch your container instances with. If you created your Amazon ECS container instance role before CloudWatch metrics were available for Amazon ECS, you might need to add this permission. For information about checking your Amazon ECS container instance role and attaching the managed IAM policy for container instances, see [To check for the `ecsInstanceRole` in the IAM console \(p. 696\)](#).

Note

You can disable CloudWatch metrics collection by setting `ECS_DISABLE_METRICS=true` in your Amazon ECS container agent configuration. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

Available metrics and dimensions

The following sections list the metrics and dimensions that Amazon ECS sends to Amazon CloudWatch.

Amazon ECS metrics

Amazon ECS provides metrics for you to monitor your resources. You can measure the CPU and memory reservation and utilization across your cluster as a whole, and the CPU and memory utilization on the services in your clusters. For your GPU workloads, you can measure your GPU reservation across your cluster.

The metrics made available will depend on the launch type of the tasks and services in your clusters. If you're using the Fargate launch type for your services, CPU and memory utilization metrics are provided to assist in the monitoring of your services. For the EC2 launch type, you will own and need to monitor the Amazon EC2 instances that make up your underlying infrastructure. Accordingly, additional CPU, memory, and GPU reservation and CPU and memory utilization metrics are made available at the cluster, service, and task level.

Amazon ECS sends the following metrics to CloudWatch every minute. When Amazon ECS collects metrics, it collects multiple data points every minute. It then aggregates them to one data point before sending the data to CloudWatch. So in CloudWatch, one sample count is actually the aggregate of multiple data points during one minute.

The `AWS/ECS` namespace includes the following metrics.

CPUReservation

The percentage of CPU units that are reserved by running tasks in the cluster.

Cluster CPU reservation (this metric can only be filtered by `ClusterName`) is measured as the total CPU units that are reserved by Amazon ECS tasks on the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster. Only container instances in `ACTIVE` or `DRAINING` status will affect CPU reservation metrics. This metric is only used for tasks using the EC2 launch type.

Valid dimensions: `ClusterName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

CPUUtilization

The percentage of CPU units that are used in the cluster or service.

Cluster CPU utilization (metrics that are filtered by `ClusterName` without `ServiceName`) is measured as the total CPU units in use by Amazon ECS tasks on the cluster, divided by the total CPU units that were registered for all of the container instances in the cluster. Only container instances in `ACTIVE` or `DRAINING` status will affect CPU utilization metrics. Cluster CPU utilization metrics are only used for tasks using the EC2 launch type.

Service CPU utilization (metrics that are filtered by `ClusterName` and `ServiceName`) is measured as the total CPU units in use by the tasks that belong to the service, divided by the total number of CPU units that are reserved for the tasks that belong to the service. Service CPU utilization metrics are used for tasks using both the Fargate and the EC2 launch type.

Valid dimensions: `ClusterName`, `ServiceName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

MemoryReservation

The percentage of memory that is reserved by running tasks in the cluster.

Cluster memory reservation (this metric can only be filtered by `ClusterName`) is measured as the total memory that is reserved by Amazon ECS tasks on the cluster, divided by the total amount of memory that was registered for all of the container instances in the cluster. Only container instances in `ACTIVE` or `DRAINING` status will affect memory reservation metrics. This metric is only used for tasks using the EC2 launch type.

Valid dimensions: `ClusterName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

MemoryUtilization

The percentage of memory that is used in the cluster or service.

Cluster memory utilization (metrics that are filtered by `ClusterName` without `ServiceName`) is measured as the total memory in use by Amazon ECS tasks on the cluster, divided by the total amount of memory that was registered for all of the container instances in the cluster. Only container instances in `ACTIVE` or `DRAINING` status will affect memory utilization metrics. Cluster memory utilization metrics are only used for tasks using the EC2 launch type.

Service memory utilization (metrics that are filtered by `ClusterName` and `ServiceName`) is measured as the total memory in use by the tasks that belong to the service, divided by the total memory that is reserved for the tasks that belong to the service. Service memory utilization metrics are used for tasks using both the Fargate and EC2 launch types.

Valid dimensions: `ClusterName`, `ServiceName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

GPUReservation

The percentage of total available GPUs that are reserved by running tasks in the cluster.

Cluster GPU reservation is measured as the number of GPUs reserved by Amazon ECS tasks on the cluster, divided by the total number of GPUs that was available on all of the GPU-enabled container instances in the cluster. Only container instances in `ACTIVE` or `DRAINING` status will affect GPU reservation metrics.

Valid dimensions: `ClusterName`.

Valid statistics: Average, Minimum, Maximum, Sum, Sample Count. The most useful statistic is Average.

Unit: Percent.

Note

If you're using tasks with the EC2 launch type and have Linux container instances, the Amazon ECS container agent relies on Docker stats metrics to gather CPU and memory data for each container running on the instance. For burstable performance instances (T3, T3a, and T2 instances), the CPU utilization metric may reflect different data compared to instance-level CPU metrics.

Dimensions for Amazon ECS metrics

Amazon ECS metrics use the `AWS/ECS` namespace and provide metrics for the following dimensions. Metrics for a dimension only reflect the resources with running tasks during a period. For example, if you have a cluster with one service in it but that service has no tasks in a `RUNNING` state, there will be no metrics sent to CloudWatch. If you have two services and one of them has running tasks and the other doesn't, only the metrics for the service with running tasks would be sent.

`ClusterName`

This dimension filters the data that you request for all resources in a specified cluster. All Amazon ECS metrics are filtered by `ClusterName`.

`ServiceName`

This dimension filters the data that you request for all resources in a specified service within a specified cluster.

Cluster reservation

Cluster reservation metrics are measured as the percentage of CPU, memory, and GPUs that are reserved by all Amazon ECS tasks on a cluster when compared to the aggregate CPU, memory, and GPUs that were registered for each active container instance in the cluster. Only container instances in `ACTIVE` or `DRAINING` status will affect cluster reservation metrics. This metric is used only on clusters with tasks or services using the EC2 launch type. It's not supported on clusters with tasks using the Fargate launch type.

$$\text{Cluster CPU reservation} = \frac{(\text{Total CPU units reserved by tasks in cluster}) \times 100}{(\text{Total CPU units registered by container instances in cluster})}$$

$$\text{Cluster memory reservation} = \frac{(\text{Total MiB of memory reserved by tasks in cluster} \times 100)}{(\text{Total MiB of memory registered by container instances in cluster})}$$

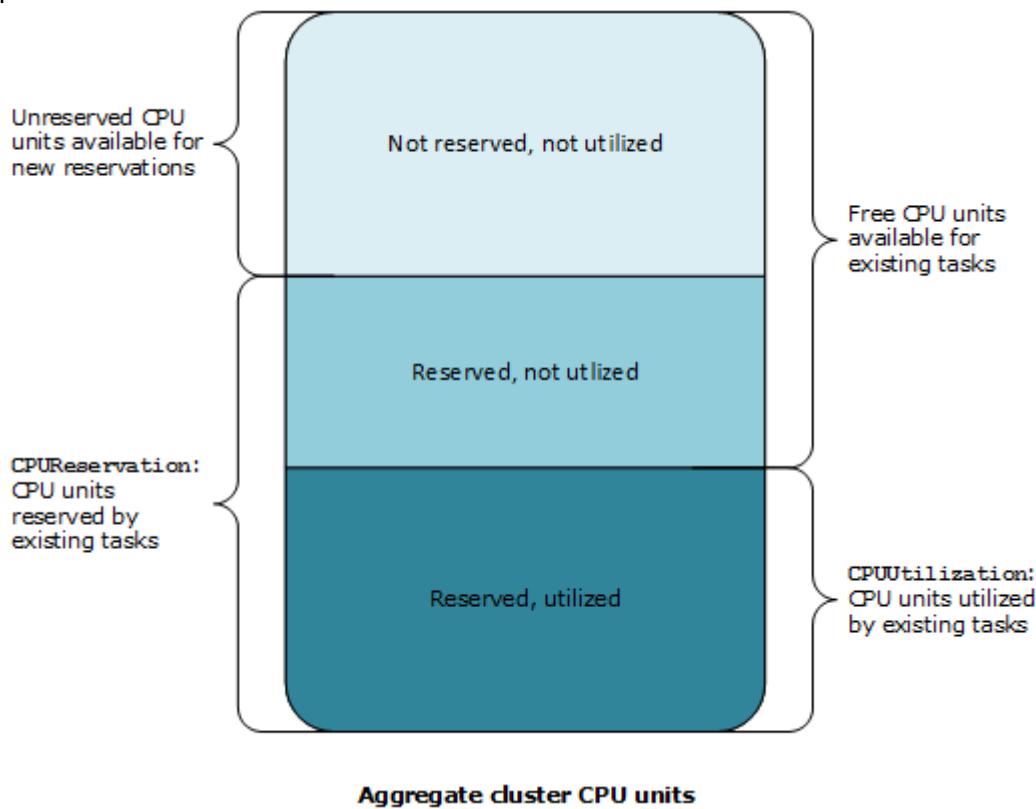
$$\text{Cluster GPU reservation} = \frac{(\text{Total GPUs reserved by tasks in cluster} \times 100)}{(\text{Total GPUs registered by container instances in cluster})}$$

When you run a task in a cluster, Amazon ECS parses its task definition and reserves the aggregate CPU units, MiB of memory, and GPUs that are specified in its container definitions. Each minute, Amazon ECS calculates the number of CPU units, MiB of memory, and GPUs that are currently reserved for each task that is running in the cluster. The total amount of CPU, memory, and GPUs reserved for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster. If you specify a soft limit (`memoryReservation`), it's used to calculate the amount of reserved memory. Otherwise, the hard limit (`memory`) is used. For more information about hard and soft limits, see [Task Definition Parameters](#).

For example, a cluster has two active container instances registered: a `c4.4xlarge` instance and a `c4.large` instance. The `c4.4xlarge` instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The `c4.large` instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If a task definition reserves 1,024 CPU units and 2,048 MiB of memory, and ten tasks are started with this task definition on this cluster (and no other tasks are currently running), a total of 10,240 CPU units and 20,480 MiB of memory are reserved. This is reported to CloudWatch as 55% CPU reservation and 60% memory reservation for the cluster.

The following illustration shows the total registered CPU units in a cluster and what their reservation and utilization means to existing tasks and new task placement. The lower (Reserved, used) and center (Reserved, not used) blocks represent the total CPU units that are reserved for the existing tasks that are running on the cluster, or the `CPUReservation` CloudWatch metric. The lower block represents the reserved CPU units that the running tasks are actually using on the cluster, or the `CPUUtilization` CloudWatch metric. The upper block represents CPU units that are not reserved by existing tasks; these CPU units are available for new task placement. Existing tasks can use these unreserved CPU units as well, if their need for CPU resources increases. For more information, see the [cpu \(p. 219\)](#) task definition parameter documentation.



Cluster utilization

Cluster utilization is measured as the percentage of CPU and memory that is used by all Amazon ECS tasks on a cluster when compared to the aggregate CPU and memory that was registered for each active container instance in the cluster. Only container instances in ACTIVE or DRAINING status will affect cluster utilization metrics. A GPU utilization metric isn't supported because it's not possible to overcommit a GPU. This metric is used only on clusters with tasks or services using the EC2 launch type. It's not supported on clusters with tasks using the Fargate launch type.

$$\text{Cluster CPU utilization} = \frac{(\text{Total CPU units used by tasks in cluster}) \times 100}{(\text{Total CPU units registered by container instances in cluster})}$$

$$(\text{Total MiB of memory used by tasks in cluster} \times 100)$$

```
Cluster memory utilization =  
-----  
          (Total MiB of memory registered by container instances in  
cluster)
```

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task that is running on that container instance, and this information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total registered resources for the cluster.

For example, a cluster has two active container instances registered, a `c4.4xlarge` instance and a `c4.1large` instance. The `c4.4xlarge` instance registers into the cluster with 16,384 CPU units and 30,158 MiB of memory. The `c4.1large` instance registers with 2,048 CPU units and 3,768 MiB of memory. The aggregate resources of this cluster are 18,432 CPU units and 33,926 MiB of memory.

If ten tasks are running on this cluster and each task consumes 1,024 CPU units and 2,048 MiB of memory, a total of 10,240 CPU units and 20,480 MiB of memory are used on the cluster. This is reported to CloudWatch as 55% CPU utilization and 60% memory utilization for the cluster.

Service utilization

Service utilization is measured as the percentage of CPU and memory that is used by the Amazon ECS tasks that belong to a service on a cluster when compared to the CPU and memory that is specified in the service's task definition. This metric is supported for services with tasks using both the EC2 and Fargate launch types.

```
Service CPU utilization =  
-----  
          (Total CPU units used by tasks in service) x 100  
          -----  
          (Total CPU units specified in task definition) x (number of  
tasks in service)
```

```
100  
Service memory utilization =  
-----  
          (Total MiB of memory used by tasks in service) x  
          -----  
          (Total MiB of memory specified in task definition) x (number  
of tasks in service)
```

Each minute, the Amazon ECS container agent on each container instance calculates the number of CPU units and MiB of memory that are currently being used for each task owned by the service that is running on that container instance, and this information is reported back to Amazon ECS. The total amount of CPU and memory used for all tasks owned by the service that are running on the cluster is calculated, and those numbers are reported to CloudWatch as a percentage of the total resources that are specified for the service in the service's task definition. If you specify a soft limit (`memoryReservation`), it's used to calculate the amount of reserved memory. Otherwise, the hard limit (`memory`) is used. For more information about hard and soft limits, see [Task Definition Parameters](#).

For example, the task definition for a service specifies a total of 512 CPU units and 1,024 MiB of memory (with the hard limit `memory` parameter) for all of its containers. The service has a desired count of 1 running task, the service is running on a cluster with 1 `c4.1large` container instance (with 2,048 CPU units and 3,768 MiB of total memory), and there are no other tasks running on the cluster. Although the task specifies 512 CPU units, because it is the only running task on a container instance with 2,048 CPU units, it can use up to four times the specified amount (2,048 / 512). However, the specified memory of 1,024 MiB is a hard limit and it can't be exceeded, so in this case, service memory utilization can't exceed 100%.

If the previous example used the soft limit `memoryReservation` instead of the hard limit `memory` parameter, the service's tasks could use more than the specified 1,024 MiB of memory as needed. In this case, the service's memory utilization could exceed 100%.

If your application has a sudden spike in memory utilization for a short amount of time, you will not see the service memory utilization increasing because Amazon ECS collects multiple data points every minute, and then aggregates them to one data point that is sent to CloudWatch.

If this task is performing CPU-intensive work during a period and using all 2,048 of the available CPU units and 512 MiB of memory, the service reports 400% CPU utilization and 50% memory utilization. If the task is idle and using 128 CPU units and 128 MiB of memory, the service reports 25% CPU utilization and 12.5% memory utilization.

Note

In this example, the CPU utilization will only go above 100% when the CPU units are defined at the container level. If you define CPU units at the task level, the utilization will not go above the defined task-level limit.

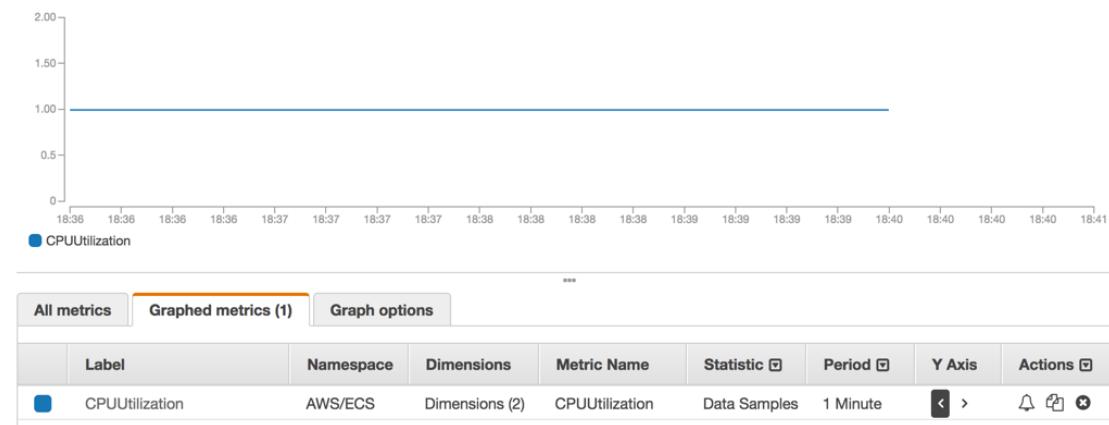
Service RUNNING task count

You can use CloudWatch metrics to view the number of tasks in your services that are in the `RUNNING` state. For example, you can set a CloudWatch alarm for this metric to alert you if the number of running tasks in your service falls below a specified value.

To view the number of running tasks in a service

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. On the navigation pane, choose **Metrics**.
3. On the **All metrics** tab, choose **ECS**.
4. Choose **ClusterName**, **ServiceName** and then choose any metric (either `CPUUtilization` or `MemoryUtilization`) that corresponds to the service to view running tasks in.
5. On the **Graphed metrics** tab, change **Period** to **1 Minute** and **Statistic** to **Sample Count**.

The value displayed in the graph indicates the number of `RUNNING` tasks in the service.



Viewing Amazon ECS metrics

After you have enabled CloudWatch metrics for Amazon ECS, you can view those metrics on the Amazon ECS and CloudWatch consoles. The Amazon ECS console provides a 24-hour maximum, minimum, and

average view of your cluster and service metrics. The CloudWatch console provides a fine-grained and customizable display of your resources, as well as the number of running tasks in a service.

Topics

- [Viewing cluster metrics using the Amazon ECS console \(p. 627\)](#)
- [Viewing service metrics using the Amazon ECS console \(p. 627\)](#)
- [Viewing Amazon ECS metrics using the CloudWatch console \(p. 627\)](#)

Viewing cluster metrics using the Amazon ECS console

Cluster and service metrics are available on the Amazon ECS console. The view provided for cluster metrics shows the average, minimum, and maximum values for the previous 24-hour period, with data points available in 5-minute intervals. For more information about cluster metrics, see [Cluster reservation \(p. 623\)](#) and [Cluster utilization \(p. 624\)](#).

To view cluster metrics on the console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. Select the cluster that you want to view metrics for.
3. On the **Cluster: *cluster-name*** page, choose **Metrics**.

Viewing service metrics 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 \(p. 625\)](#).

To view service metrics in the console

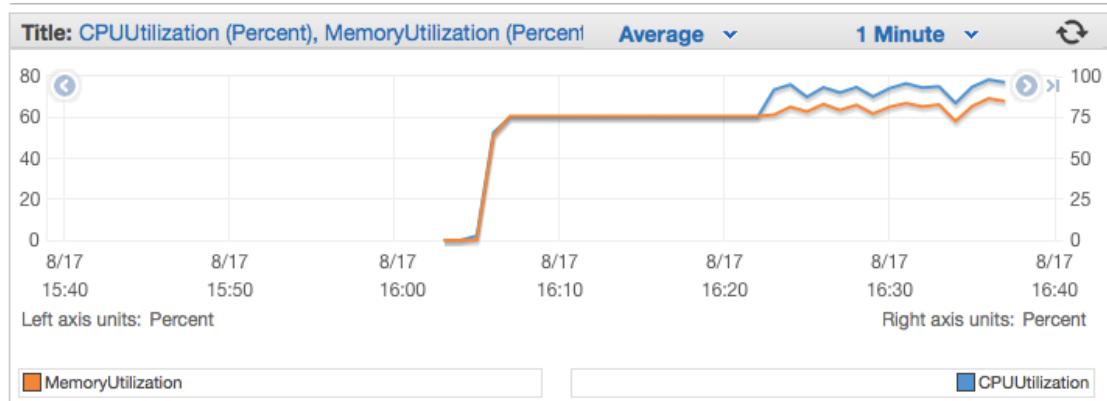
1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. Select the cluster that contains the service that you want to view metrics for.
3. On the **Cluster: *cluster-name*** page, choose **Services**.
4. Choose the service that you want to view metrics for.
5. On the **Service: *service-name*** page, choose **Metrics**.

Viewing Amazon ECS metrics using the CloudWatch console

Amazon ECS cluster and service metrics can also be viewed on the CloudWatch console. The console provides the most detailed view of Amazon ECS metrics, and you can tailor the views to suit your needs. You can view [Cluster reservation \(p. 623\)](#), [Cluster utilization \(p. 624\)](#), [Service utilization \(p. 625\)](#), and the [Service RUNNING task count \(p. 626\)](#). For more information, see the [Amazon CloudWatch User Guide](#).

To view metrics in the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the **Metrics** section in the navigation pane, choose **ECS**.
3. Choose the metrics to view. Cluster metrics are scoped as **ECS > ClusterName** and service utilization metrics are scoped as **ECS > ClusterName, ServiceName**. The following example shows cluster CPU and memory utilization.



Tutorial: Scaling container instances with CloudWatch alarms

Note

In December 2019, Amazon ECS launched cluster auto scaling, as an alternative method for scaling container instances. For more information, see [Amazon ECS cluster auto scaling \(p. 189\)](#).

The following procedures help you to create an Auto Scaling group for an Amazon ECS cluster. The Auto Scaling group contains container instances that you can scale out (and in) using CloudWatch alarms.

Depending on the Amazon EC2 instance types that you use in your clusters, and quantity of container instances that you have in a cluster, your tasks have a limited amount of resources that they can use while running. Amazon ECS monitors the resources available in the cluster to work with the schedulers to place tasks. If your cluster runs low on any of these resources, such as memory, you are eventually unable to launch more tasks until you add more container instances, reduce the number of desired tasks in a service, or stop some of the running tasks in your cluster to free up the constrained resource.

In this tutorial, you create a CloudWatch alarm and a step scaling policy using the `MemoryReservation` metric for your cluster. When the memory reservation of your cluster rises above 75% (meaning that only 25% of the memory in your cluster is available for new tasks to reserve), the alarm triggers the Auto Scaling group to add another instance and provide more resources for your tasks and services.

Prerequisites

This tutorial assumes that you have enabled CloudWatch metrics for your clusters and services. Metrics are not available until the clusters and services send the metrics to CloudWatch, and you cannot create CloudWatch alarms for metrics that do not exist yet. For more information, see [Enabling CloudWatch metrics \(p. 620\)](#).

Step 1: Create a CloudWatch alarm for a metric

After you have enabled CloudWatch metrics for your clusters and services, and the metrics for your cluster are visible in the CloudWatch console, you can set alarms on the metrics. For more information, see [Creating Amazon CloudWatch Alarms in the Amazon CloudWatch User Guide](#).

For this tutorial, you create an alarm on the cluster `MemoryReservation` metric to alert when the cluster's memory reservation is above 75%.

To create a CloudWatch alarm on a metric

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. On the left navigation, choose **Alarms, Create Alarm**.
3. In the **CloudWatch Metrics by Category** section, choose **ECS Metrics > ClusterName**.
4. On the **Modify Alarm** page, choose the **MemoryReservation** metric for the default cluster and choose **Next**.
5. In the **Alarm Threshold** section, enter a name and description for your alarm.
 - **Name:** memory-above-75-pct
 - **Description:** Cluster memory reservation above 75%
6. Set the threshold and time period requirement to **MemoryReservation** greater than 75% for 1 period.

Alarm Threshold

Provide the details and threshold for your alarm. Use the graph on the right to help set the appropriate threshold.

Name:

Description:

Whenever: MemoryReservation

is:

for: consecutive period(s)

7. (Optional) Configure a notification to send when the alarm is triggered. You can also choose to delete the notification if you don't want to configure one now.
8. Choose **Create Alarm**. Now you can use this alarm to trigger your Auto Scaling group to add a container instance when the memory reservation is above 75%.
9. (Optional) You can also create another alarm that triggers when the memory reservation is below 25%, which you can use to remove a container instance from your Auto Scaling group.

Step 2: Create a launch configuration for an Auto Scaling group

Now that you have enabled CloudWatch metrics and created an alarm based on one of those metrics, you can create a launch configuration and an Auto Scaling group for your cluster. For more information and other configuration options, see [Launch Configurations](#) in the *Amazon EC2 Auto Scaling User Guide*.

To create an Auto Scaling launch configuration

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the left navigation pane, choose **Auto Scaling Groups**.
3. On the **Welcome to Auto Scaling** page, choose **Create Auto Scaling Group**.
4. On the **Create Auto Scaling Group** page, choose **Create a new launch configuration**.
5. On the **Choose AMI** step of the **Create Auto Scaling Group** wizard, choose **Community AMIs**.
6. Choose the latest Amazon ECS-optimized Amazon Linux 2 AMI for your Auto Scaling group. For information on how to retrieve the latest Amazon ECS-optimized Amazon Linux 2 AMI, see [Retrieving Amazon ECS-Optimized AMI metadata \(p. 339\)](#).
7. On the **Choose Instance Type** step of the **Create Auto Scaling Group** wizard, choose an instance type for your Auto Scaling group and choose **Next: Configure details**.
8. On the **Configure details** step of the **Create Auto Scaling Group** wizard, enter the following information. The other fields are optional. For more information, see [Creating Launch Configurations](#) in the *Amazon EC2 Auto Scaling User Guide*.

- **Name:** Enter a name for your launch configuration.
 - **IAM role:** Select the `ecsInstanceRole` for your container instances. If you do not have this role configured, see [Amazon ECS container instance IAM role \(p. 695\)](#).
 - **IP Address Type:** Select the IP address type option for your container instances. To allow external traffic to be able to reach your containers, choose **Assign a public IP address to every instance**.
9. Expand the **Advanced Details** section to specify user data for your Amazon ECS container instances. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

Paste the following script into the **User data** field. Reference the cluster name that you are working with.

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

10. Choose **Next: Add Storage**.
11. On the **Add Storage** step of the **Create Auto Scaling Group** wizard, make any storage configuration changes needed for your instances and choose **Next: Configure Security Group**.
12. On the **Configure Security Group** step of the **Create Auto Scaling Group** wizard, select an existing security group that meets the needs of your containers, or create a new security group, and choose **Review**.
13. Review your launch configuration and choose **Create launch configuration**.
14. Select a private key to use for connecting to your instances with SSH and choose **Create launch configuration**. Move on to creating an Auto Scaling group with your new launch configuration.

Step 3: Create an Auto Scaling group with step scaling policies

After the launch configuration is complete, continue with the following procedure to create an Auto Scaling group that uses your launch configuration.

To create an Auto Scaling group with step scaling policies

1. On the **Configure Auto Scaling group details** step of the **Create Auto Scaling Group** wizard, enter the following information and then choose **Next: Configure scaling policies**:
 - **Group name:** Enter a name for your Auto Scaling group.
 - **Group size:** Specify the number of container instances with which your Auto Scaling group should start.
 - **Network:** Select a VPC into which to launch your container instances.
 - **Subnet:** Select the subnets into which to launch your container instances. For a highly available cluster, we recommend that you enable all of the subnets in the Region.
2. On the **Configure scaling policies** step of the **Create Auto Scaling Group** wizard, choose **Use scaling policies to adjust the capacity of this group**.
3. Enter the minimum and maximum number of container instances for your Auto Scaling group.
4. Choose **Scale the Auto Scaling group using step or simple scaling policies**.
5. In the **Increase Group Size** section, enter the following information:
 - **Execute policy when:** Select the `memory-above-75-pct` CloudWatch alarm that you configured earlier.
 - **Take the action:** Enter the number of capacity units (instances) to add to your cluster when the alarm is triggered.
6. If you configured an alarm to trigger a group size reduction, set that alarm in the **Decrease Group Size** section and specify how many instances to remove if that alarm is triggered. Otherwise,

collapse the **Decrease Group Size** section by choosing the X in the upper-right-hand corner of the section.

Note

If you configure your Auto Scaling group to remove container instances, any tasks running on the removed container instances are stopped. If your tasks are running as part of a service, Amazon ECS restarts those tasks on another instance if the required resources are available (CPU, memory, ports). However, tasks that were started manually are not restarted automatically.

7. Choose **Review, Create Auto Scaling Group**.

Step 4: Verify and test your Auto Scaling group

Now that you've created your Auto Scaling group, you should see your instances launching in the Amazon EC2 console **Instances** page. These instances should register into your ECS cluster as well after they launch.

Verify that the EC2 instances are registered with the cluster. From the ECS console, select the cluster that you registered your instances with. On the **Cluster** page, choose **ECS Instances**. Verify that the **Agent Connected** value is **True** for the instances displayed.

To test that your Auto Scaling group is configured properly, create some tasks that consume a considerable amount of memory and start launching them into your cluster. After your cluster exceeds the 75% memory reservation from the CloudWatch alarm for the specified number of periods, you should see a new instance launch in the Amazon EC2 console.

Step 5: Cleaning up

After you no longer need a step scaling policy, you can delete it. You also need to delete the CloudWatch alarms. Deleting a step scaling policy deletes the underlying alarm action, but does not delete the CloudWatch alarm associated with the scaling policy, even if it no longer has an associated action.

To delete a step scaling policy and its associated CloudWatch alarm

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups**.
3. Select the Auto Scaling group.
4. On the **Scaling Policies** tab, choose **Actions, Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.
6. Do the following to delete the CloudWatch alarm that was associated with the policy.
 - a. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
 - b. On the navigation pane, choose **Alarms**.
 - c. Choose the alarm and choose **Action, Delete**.
 - d. When prompted for confirmation, choose **Delete**.

When you have completed this tutorial, you may choose to keep your Auto Scaling group and Amazon EC2 instances in service for your cluster. However, if you are not actively using these resources, you should consider cleaning them up so your account does not incur unnecessary charges. You can delete your Auto Scaling group to terminate the Amazon EC2 instances within it, but your launch configuration remains intact. You can create a new Auto Scaling group with the launch configuration later, if you choose.

To delete your Auto Scaling group

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the left navigation pane, choose **Auto Scaling Groups**.
3. Choose the Auto Scaling group that you created earlier.
4. Choose **Actions, Delete**.
5. Choose **Yes, Delete**.

Amazon ECS events and EventBridge

Amazon EventBridge enables you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Adding events to log groups in CloudWatch Logs
- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon Simple Queue Service (Amazon SQS) queue

For more information, see [Getting Started with Amazon EventBridge](#) in the *Amazon EventBridge User Guide*.

You can use Amazon ECS events for EventBridge to receive near real-time notifications regarding the current state of your Amazon ECS clusters. If your tasks are using the Fargate launch type, you can see the state of your tasks. If your tasks are using the EC2 launch type, you can see the state of both the container instances and the current state of all tasks running on those container instances. For services, you can see events related to the health of your service.

Using EventBridge, you can build custom schedulers on top of Amazon ECS that are responsible for orchestrating tasks across clusters and monitoring the state of clusters in near real time. You can eliminate scheduling and monitoring code that continuously polls the Amazon ECS service for status changes and instead handle Amazon ECS state changes asynchronously using any EventBridge target. Targets might include AWS Lambda, Amazon Simple Queue Service, Amazon Simple Notification Service, or Amazon Kinesis Data Streams.

An Amazon ECS event stream ensures that every event is delivered at least one time. If duplicate events are sent, the event provides enough information to identify duplicates. For more information, see [Handling events \(p. 644\)](#).

Events are relatively ordered, so that you can easily tell when an event occurred in relation to other events.

Topics

- [Amazon ECS events \(p. 633\)](#)
- [Handling events \(p. 644\)](#)

Amazon ECS events

Amazon ECS sends three types of events to EventBridge: container instance state change events, task state change events, and service action events. If these resources change, an event is triggered. These events and their possible causes are described in greater detail in the following sections.

Note

Amazon ECS may add other event types, sources, and details in the future. If you are programmatically deserializing event JSON data, make sure that your application is prepared to handle unknown properties to avoid issues if and when these additional properties are added.

In some cases, multiple events are triggered for the same activity. For example, when a task is started on a container instance, a task state change event is triggered for the new task. A container instance state change event is triggered to account for the change in available resources, such as CPU, memory, and available ports, on the container instance. Likewise, if a container instance is terminated, events are triggered for the container instance, the container agent connection status, and every task that was running on the container instance.

Container state change and task state change events contain two `version` fields: one in the main body of the event, and one in the `detail` object of the event. The following describes the differences between these two fields:

- The `version` field in the main body of the event is set to 0 on all events. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.
- The `version` field in the `detail` object of the event describes the version of the associated resource. Each time a resource changes state, this version is incremented. Because events can be sent multiple times, this field allows you to identify duplicate events. Duplicate events have the same version in the `detail` object. If you are replicating your Amazon ECS container instance and task state with EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the version reported in EventBridge for the resource (inside the `detail` object) to verify that the version in your event stream is current.

Service action events only contain the `version` field in the main body.

Container instance state change events

The following scenarios trigger container instance state change events:

You call the `StartTask`, `RunTask`, or `StopTask` API operations, either directly or with the AWS Management Console or SDKs.

Placing or stopping tasks on a container instance modifies the available resources on the container instance, such as CPU, memory, and available ports.

The Amazon ECS service scheduler starts or stops a task.

Placing or stopping tasks on a container instance modifies the available resources on the container instance, such as CPU, memory, and available ports.

The Amazon ECS container agent calls the `SubmitTaskStateChange` API operation with a `STOPPED` status for a task with a desired status of `RUNNING`.

The Amazon ECS container agent monitors the state of tasks on your container instances, and it reports any state changes. If a task that is supposed to be `RUNNING` is transitioned to `STOPPED`, the agent releases the resources that were allocated to the stopped task, such as CPU, memory, and available ports.

You deregister the container instance with the `DeregisterContainerInstance` API operation, either directly or with the AWS Management Console or SDKs.

Deregistering a container instance changes the status of the container instance and the connection status of the Amazon ECS container agent.

A task was stopped when an EC2 instance was stopped.

When you stop a container instance, the tasks that are running on it are transitioned to the `STOPPED` status.

The Amazon ECS container agent registers a container instance for the first time.

The first time the Amazon ECS container agent registers a container instance (at launch or when first run manually), this creates a state change event for the instance.

The Amazon ECS container agent connects or disconnects from Amazon ECS.

When the Amazon ECS container agent connects or disconnects from the Amazon ECS backend, it changes the `agentConnected` status of the container instance.

Note

The Amazon ECS container agent disconnects and reconnects several times per hour as a part of its normal operation, so agent connection events should be expected. These events are not an indication that there is an issue with the container agent or your container instance.

You upgrade the Amazon ECS container agent on an instance.

The container instance detail contains an object for the container agent version. If you upgrade the agent, this version information changes and triggers an event.

Example Container instance state change event

Container instance state change events are delivered in the following format. The `detail` section below resembles the `ContainerInstance` object that is returned from a `DescribeContainerInstances` API operation in the *Amazon Elastic Container Service API Reference*. For more information about EventBridge parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

```
{  
    "version": "0",  
    "id": "8952ba83-7be2-4ab5-9c32-6687532d15a2",  
    "detail-type": "ECS Container Instance State Change",  
    "source": "aws.ecs",  
    "account": "111122223333",  
    "time": "2016-12-06T16:41:06Z",  
    "region": "us-east-1",  
    "resources": [  
        "arn:aws:ecs:us-east-1:111122223333:container-instance/  
b54a2a04-046f-4331-9d74-3f6d7f6ca315"  
    ],  
    "detail": {  
        "agentConnected": true,  
        "attributes": [  
            {  
                "name": "com.amazonaws.ecs.capability.logging-driver.syslog"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.task-iam-role-network-host"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"  
            },  
            {  
                "name": "com.amazonaws.ecs.capability.logging-driver.json-file"  
            }  
        ]  
    }  
}
```

```

},
{
  "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
},
{
  "name": "com.amazonaws.ecs.capability.privileged-container"
},
{
  "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
},
{
  "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
},
{
  "name": "com.amazonaws.ecs.capability.ecr-auth"
},
{
  "name": "com.amazonaws.ecs.capability.docker-remote-api.1.20"
},
{
  "name": "com.amazonaws.ecs.capability.docker-remote-api.1.21"
},
{
  "name": "com.amazonaws.ecs.capability.docker-remote-api.1.22"
},
{
  "name": "com.amazonaws.ecs.capability.docker-remote-api.1.23"
},
{
  "name": "com.amazonaws.ecs.capability.task-iam-role"
}
],
"clusterArn": "arn:aws:ecs:us-east-1:111122223333:cluster/default",
"containerInstanceArn": "arn:aws:ecs:us-east-1:111122223333:container-instance/b54a2a04-046f-4331-9d74-3f6d7f6ca315",
"ec2InstanceId": "i-f3a8506b",
"registeredResources": [
  {
    "name": "CPU",
    "type": "INTEGER",
    "integerValue": 2048
  },
  {
    "name": "MEMORY",
    "type": "INTEGER",
    "integerValue": 3767
  },
  {
    "name": "PORTS",
    "type": "STRINGSET",
    "stringSetValue": [
      "22",
      "2376",
      "2375",
      "51678",
      "51679"
    ]
  },
  {
    "name": "PORTS_UDP",
    "type": "STRINGSET",
    "stringSetValue": []
  }
],
"remainingResources": [
  {

```

```

        "name": "CPU",
        "type": "INTEGER",
        "integerValue": 1988
    },
    {
        "name": "MEMORY",
        "type": "INTEGER",
        "integerValue": 767
    },
    {
        "name": "PORTS",
        "type": "STRINGSET",
        "stringSetValue": [
            "22",
            "2376",
            "2375",
            "51678",
            "51679"
        ]
    },
    {
        "name": "PORTS_UDP",
        "type": "STRINGSET",
        "stringSetValue": []
    }
],
"status": "ACTIVE",
"version": 14801,
"versionInfo": {
    "agentHash": "aebcbca",
    "agentVersion": "1.13.0",
    "dockerVersion": "DockerVersion: 1.11.2"
},
"updatedAt": "2016-12-06T16:41:06.991Z"
}
}

```

Task state change events

The following scenarios trigger task state change events:

You call the `StartTask`, `RunTask`, or `StopTask` API operations, either directly or with the AWS Management Console, AWS CLI, or SDKs.

Starting or stopping tasks creates new task resources or modifies the state of existing task resources. The Amazon ECS service scheduler starts or stops a task.

Starting or stopping tasks creates new task resources or modifies the state of existing task resources. The Amazon ECS container agent calls the `SubmitTaskStateChange` API operation.

The Amazon ECS container agent monitors the state of tasks on your container instances, and it reports any state changes. State changes might include changes from `PENDING` to `RUNNING` or from `RUNNING` to `STOPPED`.

You force deregistration of the underlying container instance with the `DeregisterContainerInstance` API operation and the `force` flag, either directly or with the AWS Management Console or SDKs.

Deregistering a container instance changes the status of the container instance and the connection status of the Amazon ECS container agent. If tasks are running on the container instance, the `force` flag must be set to allow deregistration. This stops all tasks on the instance.

The underlying container instance is stopped or terminated.

When you stop or terminate a container instance, the tasks that are running on it are transitioned to the STOPPED status.

A container in the task changes state.

The Amazon ECS container agent monitors the state of containers within tasks. For example, if a container that is running within a task stops, this container state change triggers an event.

A task using the Fargate Spot capacity provider receives a termination notice.

When a task is using the FARGATE_SPOT capacity provider and is stopped due to a Spot interruption, a task state change event is triggered.

Example Task state change event

Task state change events are delivered in the following format. The detail section below resembles the **Task** object that is returned from a [DescribeTasks](#) API operation in the *Amazon Elastic Container Service API Reference*. If your containers are using an image hosted with Amazon ECR, the `imageDigest` field is returned.

Note

The values for the `createdAt`, `connectivityAt`, `pullStartedAt`, `startedAt`, `pullStoppedAt`, and `updatedAt` fields are UNIX timestamps in the response of a `DescribeTasks` action whereas in the task state change event they are ISO string timestamps.

For more information about CloudWatch Events parameters, see [Events and Event Patterns](#) in the *Amazon EventBridge User Guide*.

```
{  
    "version": "0",  
    "id": "3317b2af-7005-947d-b652-f55e762e571a",  
    "detail-type": "ECS Task State Change",  
    "source": "aws.ecs",  
    "account": "111122223333",  
    "time": "2020-01-23T17:57:58Z",  
    "region": "us-west-2",  
    "resources": [  
        "arn:aws:ecs:us-west-2:111122223333:task/FargateCluster/  
c13b4cb40f1f4fe4a2971f76ae5a47ad"  
    ],  
    "detail": {  
        "attachments": [  
            {  
                "id": "1789bcae-ddfb-4d10-8ebe-8ac87ddba5b8",  
                "type": "eni",  
                "status": "ATTACHED",  
                "details": [  
                    {  
                        "name": "subnetId",  
                        "value": "subnet-abcd1234"  
                    },  
                    {  
                        "name": "networkInterfaceId",  
                        "value": "eni-abcd1234"  
                    },  
                    {  
                        "name": "macAddress",  
                        "value": "0a:98:eb:a7:29:ba"  
                    },  
                    {  
                        "name": "privateIPv4Address",  
                        "value": "172.31.1.10"  
                    }  
                ]  
            }  
        ]  
    }  
}
```

```

                "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.

`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 \(p. 325\)](#). 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"
    }
}

```

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-e0e3aEXAMPLE",
    "detail-type": "ECS Deployment State Change",
    "source": "aws.ecs",
    "account": "111122223333",
    "time": "2020-05-23T12:31:14Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ecs:us-west-2:111122223333:service/default/servicetest"
    ],
    "detail": {
        "eventType": "INFO",
        "eventName": "SERVICE_DEPLOYMENT_IN_PROGRESS",
        "deploymentId": "ecs-svc/123",
        "updatedAt": "2020-05-23T11:11:11Z",
        "reason": "ECS deployment circuit breaker: rolling back to deploymentId deploymentID."
    }
}
```

Example service deployment completed event

Service deployment completed state events are delivered in the following format. For more information, see [Rolling update \(p. 564\)](#).

```
{
    "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 \(p. 564\)](#).

```
{
    "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 Amazon ECS container instance and task state with EventBridge, you can compare the version of a resource reported by the Amazon ECS APIs with the `version` reported in EventBridge for the resource to verify that the version in your event stream is current. Events with a higher `version` property number should be treated as occurring later than events with lower `version` numbers.

Example: Handling events in an AWS Lambda function

The following example shows a Lambda function written in Python 2.7 that captures both task and container instance state change events and saves them to one of two Amazon DynamoDB tables:

- `ECSCtrlInstanceState` – Stores the latest state for a container instance. The table ID is the `containerInstanceArn` value of the container instance.
- `ECSTaskState` – Stores the latest state for a task. The table ID is the `taskArn` value of the task.

```

import json
import boto3

def lambda_handler(event, context):
    id_name = ""
    new_record = {}

    # For debugging so you can see raw event format.
    print('Here is the event:')
    print(json.dumps(event))

    if event["source"] != "aws.ecs":
        raise ValueError("Function only supports input from events with a source type of: aws.ecs")

    # Switch on task/container events.
    table_name = ""
    if event["detail-type"] == "ECS Task State Change":
        table_name = "ECSTaskState"
        id_name = "taskArn"
        event_id = event["detail"]["taskArn"]
    elif event["detail-type"] == "ECS Container Instance State Change":
        table_name = "ECSCtrlInstanceState"
        id_name = "containerInstanceArn"

```

```

        event_id = event["detail"]["containerInstanceArn"]
    else:
        raise ValueError("detail-type for event is not a supported type. Exiting without
saving event.")

    new_record["cw_version"] = event["version"]
    new_record.update(event["detail"])

    # "status" is a reserved word in DDB, but it appears in containerPort
    # state change messages.
    if "status" in event:
        new_record["current_status"] = event["status"]
        new_record.pop("status")

    # Look first to see if you have received a newer version of an event ID.
    # If the version is OLDER than what you have on file, do not process it.
    # Otherwise, update the associated record with this latest information.
    print("Looking for recent event with same ID...")
    dynamodb = boto3.resource("dynamodb", region_name="us-east-1")
    table = dynamodb.Table(table_name)
    saved_event = table.get_item(
        Key={
            id_name : event_id
        }
    )
    if "Item" in saved_event:
        # Compare events and reconcile.
        print("EXISTING EVENT DETECTED: Id " + event_id + " - reconciling")
        if saved_event["Item"]["version"] < event["detail"]["version"]:
            print("Received event is a more recent version than the stored event -
updating")
            table.put_item(
                Item=new_record
            )
        else:
            print("Received event is an older version than the stored event - ignoring")
    else:
        print("Saving new event - ID " + event_id)

    table.put_item(
        Item=new_record
    )

```

Amazon ECS CloudWatch Container Insights

CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. The metrics include utilization for resources such as CPU, memory, disk, and network. The metrics are available in CloudWatch automatic dashboards. For a full list of Amazon ECS Container Insights metrics, see [Amazon ECS Container Insights Metrics](#) in the *Amazon CloudWatch User Guide*.

Operational data is collected as performance log events. These are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates higher-level aggregated metrics at the cluster, service, and task level as CloudWatch metrics. For more information, see [Container Insights Structured Logs for Amazon ECS](#) in the *Amazon CloudWatch User Guide*.

Important

Metrics collected by CloudWatch Container Insights are charged as custom metrics. For more information about CloudWatch pricing, see [CloudWatch Pricing](#). Amazon ECS also provides

monitoring metrics that are provided at no additional cost. For more information, see [Amazon ECS CloudWatch metrics \(p. 620\)](#).

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

Important

For clusters containing tasks or services using the EC2 launch type, your container instances must be running version 1.29.0 or later of the Amazon ECS agent. For more information, see [Amazon ECS container agent versions \(p. 440\)](#).

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
```

Container instance health

Amazon ECS provides container instance health monitoring. You can quickly determine whether Amazon ECS has detected any problems that might prevent your container instances from running containers. Amazon ECS performs automated checks on every running container instance with agent version 1.57.0 or later to identify issues. For more information on verifying the agent version on a container instance, see [Updating the Amazon ECS container agent \(p. 448\)](#).

Status checks are performed about twice per minute, returning a pass or a fail status. If all checks pass, the overall status of the instance is **OK**. If one or more checks fail, the overall status is **IMPAIRED**. Status checks are built into Amazon ECS container agent, so they cannot be disabled or deleted. You can view the results of these status checks to identify specific and detectable problems.

The container instance health status can be retrieved using the `DescribeContainerInstances` API. The following AWS CLI command retrieves the container instance health status.

```
aws ecs describe-container-instances \
--cluster cluster_name \
--container-instances 47279cd2cadb41cbaef2dcEXAMPLE \
--include CONTAINER_INSTANCE_HEALTH
```

The following is an example of the health status object in the output.

```
"healthStatus": {
  "overallStatus": "OK",
  "details": [
    {
      "type": "CONTAINER_RUNTIME",
      "status": "OK",
      "lastUpdated": "2021-11-10T03:30:26+00:00",
      "lastStatusChange": "2021-11-10T03:26:41+00:00"
    }
}
```

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

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 \(p. 691\)](#).

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:PutTelemetryRecords"
      ]
    }
  ]
}
```

```
        "xray:GetSamplingTargets",
        "xray:GetSamplingStatisticSummaries",
        "ssm:GetParameters"
    ],
    "Resource": "*"
}
}
```

4. (Optional) Add one or more tags to the policy, then choose **Next: Review**.
5. For **Name**, specify **AWSxDistroOpenTelemetryPolicy**.
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 **AWSxDistroOpenTelemetryPolicy**, 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 **AWSOTTaskRole** 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. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

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/AWSOpenTelemetryTaskRole",
  "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:v0.14.0",
        "logConfiguration": {
          "logDriver": "awslogs",
          "options": {
            "awslogs-create-group": "true",
            "awslogs-group": "/ecs/aws-otel-collector",
            "awslogs-region": "us-east-1",
            "awslogs-stream-prefix": "ecs"
          }
        }
      }
    }
  ]
}
```

```
"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"
}
```

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 \(p. 653\)](#)
- [Logging and Monitoring in Amazon Elastic Container Service \(p. 712\)](#)
- [Compliance Validation for Amazon Elastic Container Service \(p. 713\)](#)
- [Infrastructure Security in Amazon Elastic Container Service \(p. 714\)](#)

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 \(p. 654\)](#)
- [Authenticating With Identities \(p. 654\)](#)
- [Managing access using policies \(p. 656\)](#)
- [How Amazon Elastic Container Service works with IAM \(p. 658\)](#)
- [Amazon Elastic Container Service identity-based policy examples \(p. 663\)](#)

- [AWS managed policies for Amazon Elastic Container Service \(p. 674\)](#)
- [Service-linked role for Amazon ECS \(p. 685\)](#)
- [Amazon ECS task execution IAM role \(p. 691\)](#)
- [Amazon ECS container instance IAM role \(p. 695\)](#)
- [ECS Anywhere IAM role \(p. 697\)](#)
- [IAM Roles for Tasks \(p. 699\)](#)
- [Amazon ECS CodeDeploy IAM Role \(p. 704\)](#)
- [Amazon ECS CloudWatch Events IAM Role \(p. 708\)](#)
- [Troubleshooting Amazon Elastic Container Service identity and access \(p. 710\)](#)

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 \(p. 710\)](#).

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 \(p. 658\)](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon ECS. To view example Amazon ECS identity-based policies that you can use in IAM, see [Amazon Elastic Container Service identity-based policy examples \(p. 663\)](#).

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, you should understand what IAM features are available to use with Amazon ECS. To get a high-level view of how Amazon ECS and other AWS services work with IAM, see [AWS Services That Work with IAM](#) in the *IAM User Guide*.

Topics

- [Amazon ECS identity-based policies \(p. 658\)](#)
- [Amazon ECS resource-based policies \(p. 661\)](#)
- [Supported resource-level permissions for Amazon ECS API actions \(p. 662\)](#)
- [Authorization based on Amazon ECS tags \(p. 662\)](#)
- [Amazon ECS IAM roles \(p. 662\)](#)

Amazon ECS identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon ECS supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

Actions

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.

Policy actions in Amazon ECS use the following prefix before the action: `ecs:`. For example, to grant someone permission to create an Amazon ECS cluster with the Amazon ECS `CreateCluster` API operation, you include the `ecs:CreateCluster` action in their policy. Policy statements must include either an Action or NotAction element. Amazon ECS defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [  
    "ecs:action1",  
    "ecs:action2"]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "ecs:Describe*"
```

To see a list of Amazon ECS actions, see [Actions, Resources, and Condition Keys for Amazon Elastic Container Service](#) in the *IAM User Guide*

Resources

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": "*"
```

The Amazon ECS cluster resource has the following ARN:

```
arn:${Partition}:ecs:${Region}:${Account}:cluster/${clusterName}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#).

For example, to specify the `my-cluster` cluster in your statement, use the following ARN:

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/my-cluster"
```

To specify all clusters that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:ecs:us-east-1:123456789012:cluster/*"
```

Some Amazon ECS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Some Amazon ECS API actions can be performed on multiple resources. For example, multiple clusters can be referenced when calling the `DescribeClusters` API action. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
    "resource1",  
    "resource2"]
```

The following table describes the ARNs for each resource type used by the Amazon ECS API actions.

Important

The following table uses the new longer ARN format for Amazon ECS tasks, services, and container instances. If you have not opted in to the long ARN format, the ARNs will not include the cluster name. For more information, see [Amazon Resource Names \(ARNs\) and IDs \(p. 325\)](#).

Resource Type	ARN
All Amazon ECS resources	arn:aws:ecs:*
All Amazon ECS resources owned by the specified account in the specified region	arn:aws:ecs:region:account:*
Cluster	arn:aws:ecs:region:account:cluster/cluster-name
Container instance	arn:aws:ecs:region:account:container-instance/cluster-name/container-instance-id
Task definition	arn:aws:ecs:region:account:task-definition/task-definition-family-name:task-definition-revision-number
Service	arn:aws:ecs:region:account:service/cluster-name/service-name
Task	arn:aws:ecs:region:account:task/cluster-name/task-id
Container	arn:aws:ecs:region:account:container/cluster-name/task-id/container-id

To learn with which actions you can specify the ARN of each resource, see [Supported resource-level permissions for Amazon ECS API actions \(p. 662\)](#).

Condition keys

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 defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS Global Condition Context Keys](#) in the *IAM User Guide*.

Amazon ECS implements the following service-specific condition keys.

Condition Key	Description	Evaluation Types
aws:RequestTag/\${TagKey}	The context key is formatted "aws :RequestTag/ tag-key : " tag-value " where tag-key and tag-value are a tag key and value pair.	String

Condition Key	Description	Evaluation Types
	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".	
aws:ResourceTag/\${TagKey}	The context key is formatted "aws:ResourceTag/ <i>tag-key</i> ": " <i>tag-value</i> " where <i>tag-key</i> and <i>tag-value</i> are a tag key and value pair. Checks that the tag attached to the identity resource (user or role) matches the specified key name and value.	String
aws:TagKeys	This context key is formatted "aws:TagKeys": " <i>tag-key</i> " where <i>tag-key</i> 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/ <i>tag-key</i> ": " <i>tag-value</i> " where <i>tag-key</i> and <i>tag-value</i> 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": " <i>cluster-arn</i> " where <i>cluster-arn</i> is the ARN for the Amazon ECS cluster.	ARN, Null
ecs:container-instances	The context key is formatted "ecs:container-instances": " <i>container-instance-arns</i> " where <i>container-instance-arns</i> is one or more container instance ARNs.	ARN, Null
ecs:task-definition	The context key is formatted "ecs:task-definition": " <i>task-definition-arn</i> " where <i>task-definition-arn</i> is the ARN for the Amazon ECS task definition.	ARN, Null
ecs:service	The context key is formatted "ecs:service": " <i>service-arn</i> " where <i>service-arn</i> is the ARN for the Amazon ECS service.	ARN, Null

To learn with which actions and resources you can use a condition key, see [Supported resource-level permissions for Amazon ECS API actions \(p. 662\)](#).

Examples

To view examples of Amazon ECS identity-based policies, see [Amazon Elastic Container Service identity-based policy examples \(p. 663\)](#).

Amazon ECS resource-based policies

Amazon ECS does not support resource-based policies.

Supported resource-level permissions for Amazon ECS API actions

The term *resource-level permissions* refers to the ability to specify which resources users are allowed to perform actions on. Amazon ECS has partial support for resource-level permissions. This means that for certain Amazon ECS actions, you can control when users are allowed to use those actions based on conditions that have to be fulfilled, or specific resources that users are allowed to use. For example, you can grant users permission to launch instances, but only of a specific type, and only using a specific AMI.

For more information about the resources that are created or modified by the Amazon ECS actions, and the ARNs and Amazon ECS condition keys that you can use in an IAM policy statement, see [Actions, Resources, and Condition Keys for Amazon Elastic Container Service](#) in the *IAM User Guide*.

Authorization based on Amazon ECS tags

You can attach tags to Amazon ECS resources or pass tags in a request to Amazon ECS. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:RequestTag/key-name` or `aws:TagKeys` condition keys. For more information, see [Controlling Access Using Tags](#) in the *IAM User Guide*.

For more information about tagging Amazon ECS resources, see [Resources and tags \(p. 604\)](#).

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 \(p. 673\)](#).

Amazon ECS IAM roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

Using temporary credentials with Amazon ECS

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

Amazon ECS supports using temporary credentials.

Service-linked roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon ECS supports service-linked roles. For details about creating or managing Amazon ECS service-linked roles, see [Service-linked role for Amazon ECS \(p. 685\)](#).

Service roles

This feature allows a service to assume a [service role](#) on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon ECS supports service roles.

Amazon Elastic Container Service identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon ECS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating Policies on the JSON Tab](#) in the *IAM User Guide*.

Topics

- [Policy best practices \(p. 663\)](#)
- [Allow users to view their own permissions \(p. 664\)](#)
- [Amazon ECS first-run wizard permissions \(p. 664\)](#)
- [Cluster examples \(p. 668\)](#)
- [Container Instance Examples \(p. 669\)](#)
- [Task Definition Examples \(p. 670\)](#)
- [Run Task Example \(p. 670\)](#)
- [Start Task Example \(p. 671\)](#)
- [List and Describe Task Examples \(p. 671\)](#)
- [Create Service Example \(p. 672\)](#)
- [Update Service Example \(p. 673\)](#)
- [Describing Amazon ECS Services Based on Tags \(p. 673\)](#)

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon ECS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Amazon ECS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get started using permissions with AWS managed policies](#) in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant least privilege](#) in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam>ListGroupsForUser",
                "iam>ListAttachedUserPolicies",
                "iam>ListUserPolicies",
                "iam GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam GetPolicy",
                "iam>ListAttachedGroupPolicies",
                "iam>ListGroupPolicies",
                "iam>ListPolicyVersions",
                "iam>ListPolicies",
                "iam>ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon ECS first-run wizard permissions

The Amazon ECS first-run wizard 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 \(p. 674\)](#) 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",
                "cloudwatchlogs>CreateLogStream",
                "cloudwatchlogs>PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

```
"autoscaling:UpdateAutoScalingGroup",
"autoscaling>CreateAutoScalingGroup",
"autoscaling>CreateLaunchConfiguration",
"autoscaling>DeleteAutoScalingGroup",
"autoscaling>DeleteLaunchConfiguration",
"autoscaling:Describe*",
"cloudformation>CreateStack",
"cloudformation>DeleteStack",
"cloudformation:DescribeStack*",
"cloudformation:UpdateStack",
"cloudwatch:DescribeAlarms",
"cloudwatch>DeleteAlarms",
"cloudwatch:GetMetricStatistics",
"cloudwatch:PutMetricAlarm",
"codedeploy>CreateApplication",
"codedeploy>CreateDeployment",
"codedeploy>CreateDeploymentGroup",
"codedeploy>GetApplication",
"codedeploy>GetDeployment",
"codedeploy>GetDeploymentGroup",
"codedeploy>ListApplications",
"codedeploy>ListDeploymentGroups",
"codedeploy>ListDeployments",
"codedeploy>StopDeployment",
"codedeploy>GetDeploymentTarget",
"codedeploy>ListDeploymentTargets",
"codedeploy>GetDeploymentConfig",
"codedeploy>GetApplicationRevision",
"codedeploy>RegisterApplicationRevision",
"codedeploy>BatchGetApplicationRevisions",
"codedeploy>BatchGetDeploymentGroups",
"codedeploy>BatchGetDeployments",
"codedeploy>BatchGetApplications",
"codedeploy>ListApplicationRevisions",
"codedeploy>ListDeploymentConfigs",
"codedeploy>ContinueDeployment",
"sns>ListTopics",
"lambda>ListFunctions",
"ec2:AssociateRouteTable",
"ec2:AttachInternetGateway",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:CancelSpotFleetRequests",
"ec2>CreateInternetGateway",
"ec2>CreateLaunchTemplate",
"ec2>CreateRoute",
"ec2>CreateRouteTable",
"ec2>CreateSecurityGroup",
"ec2>CreateSubnet",
"ec2>CreateVpc",
"ec2>DeleteLaunchTemplate",
"ec2>DeleteSubnet",
"ec2>DeleteVpc",
"ec2:Describe*",
"ec2:DetachInternetGateway",
"ec2:DisassociateRouteTable",
"ec2:ModifySubnetAttribute",
"ec2:ModifyVpcAttribute",
"ec2:RunInstances",
"ec2:RequestSpotFleet",
"elasticloadbalancing>CreateListener",
"elasticloadbalancing>CreateLoadBalancer",
"elasticloadbalancing>CreateRule",
"elasticloadbalancing>CreateTargetGroup",
"elasticloadbalancing>DeleteListener",
"elasticloadbalancing>DeleteLoadBalancer",
"elasticloadbalancing>DeleteRule",
```

```

        "elasticloadbalancing:DeleteTargetGroup",
        "elasticloadbalancing:DescribeListeners",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeRules",
        "elasticloadbalancing:DescribeTargetGroups",
        "ecs:*",
        "events:DescribeRule",
        "events:DeleteRule",
        "events>ListRuleNamesByTarget",
        "events>ListTargetsByRule",
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "iam>ListAttachedRolePolicies",
        "iam>ListInstanceProfiles",
        "iam>ListRoles",
        "logs>CreateLogGroup",
        "logs>DescribeLogGroups",
        "logs>FilterLogEvents",
        "route53:GetHostedZone",
        "route53>ListHostedZonesByName",
        "route53>CreateHostedZone",
        "route53>DeleteHostedZone",
        "route53:GetHealthCheck",
        "servicediscovery>CreatePrivateDnsNamespace",
        "servicediscovery>CreateService",
        "servicediscovery>GetNamespace",
        "servicediscovery>GetOperation",
        "servicediscovery>GetService",
        "servicediscovery>ListNamespaces",
        "servicediscovery>ListServices",
        "servicediscovery>UpdateService",
        "servicediscovery>DeleteService"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ssm:GetParametersByPath",
        "ssm:GetParameters",
        "ssm:GetParameter"
    ],
    "Resource": "arn:aws:ssm:*::parameter/aws/service/ecs*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2>DeleteInternetGateway",
        "ec2>DeleteRoute",
        "ec2>DeleteRouteTable",
        "ec2>DeleteSecurityGroup"
    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringLike": {
            "ec2:ResourceTag/aws:cloudformation:stack-name": "EC2ContainerService-**"
        }
    }
},
{

```

```

    "Action": "iam:PassRole",
    "Effect": "Allow",
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringLike": {
            "iam:PassedToService": "ecs-tasks.amazonaws.com"
        }
    }
},
{
    "Action": "iam:PassRole",
    "Effect": "Allow",
    "Resource": [
        "arn:aws:iam::*:role/ecsInstanceRole*"
    ],
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "ec2.amazonaws.com",
                "ec2.amazonaws.com.cn"
            ]
        }
    }
},
{
    "Action": "iam:PassRole",
    "Effect": "Allow",
    "Resource": [
        "arn:aws:iam::*:role/ecsAutoscaleRole*"
    ],
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "application-autoscaling.amazonaws.com",
                "application-autoscaling.amazonaws.com.cn"
            ]
        }
    }
},
{
    "Effect": "Allow",
    "Action": "iam>CreateServiceLinkedRole",
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": [
                "ecs.amazonaws.com",
                "spot.amazonaws.com",
                "spotfleet.amazonaws.com",
                "ecs.application-autoscaling.amazonaws.com",
                "autoscaling.amazonaws.com"
            ]
        }
    }
}
]
}

```

The first run wizard also attempts to automatically create different IAM roles depending on the launch type of the tasks used. Examples are the Amazon ECS service role, container instance IAM role, and the task execution IAM role. To ensure that the first-run experience is able to create these IAM roles, one of the following must be true:

- Your user has administrator access. For more information, see [Setting up with Amazon ECS \(p. 7\)](#).
- Your user has the IAM permissions to create a service role. For more information, see [Creating a role to delegate permissions to an AWS service](#).
- You have a user with administrator access manually create the required IAM role so it is available on the account to be used. For more information, see the following:
 - [Service-linked role for Amazon ECS \(p. 685\)](#)
 - [Amazon ECS container instance IAM role \(p. 695\)](#)
 - [Amazon ECS task execution IAM role \(p. 691\)](#)

Cluster examples

The following IAM policy allows permission to create and list clusters. The `CreateCluster` and `ListClusters` actions do not accept any resources, so the resource definition is set to `*` for all resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs>ListClusters"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

The following IAM policy allows permission to describe and delete a specific cluster. The `DescribeClusters` and `DeleteCluster` actions accept cluster ARNs as resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeClusters",
        "ecs>DeleteCluster"
      ],
      "Resource": [
        "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/<cluster_name>"
      ]
    }
  ]
}
```

The following IAM policy can be attached to a user or group that would only allow that user or group to perform operations on a specific cluster.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [

```

```

        "ecs:Describe*",
        "ecs>List*"
    ],
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": [
        "ecs>DeleteCluster",
        "ecs>DeregisterContainerInstance",
        "ecs>ListContainerInstances",
        "ecs>RegisterContainerInstance",
        "ecs>SubmitContainerStateChange",
        "ecs>SubmitTaskStateChange"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
},
{
    "Action": [
        "ecs>DescribeContainerInstances",
        "ecs>DescribeTasks",
        "ecs>ListTasks",
        "ecs>UpdateContainerAgent",
        "ecs>StartTask",
        "ecs>StopTask",
        "ecs>RunTask"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
        "ArnEquals": {
            "ecs:cluster": "arn:aws:ecs:us-east-1:<aws_account_id>:cluster/default"
        }
    }
}
]
}

```

Container Instance Examples

Container instance registration is handled by the Amazon ECS agent, but there may be times where you want to allow a user to deregister an instance manually from a cluster. Perhaps the container instance was accidentally registered to the wrong cluster, or the instance was terminated with tasks still running on it.

The following IAM policy allows a user to list and deregister container instances in a specified cluster:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs>DeregisterContainerInstance",
                "ecs>ListContainerInstances"
            ],
            "Resource": [
                "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
            ]
        }
    ]
}
```

The following IAM policy allows a user to describe a specified container instance in a specified cluster. To open this permission up to all container instances in a cluster, you can replace the container instance UUID with *.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeContainerInstances"
            ],
            "Condition": {
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
                }
            },
            "Resource": [
                "arn:aws:ecs:<region>:<aws_account_id>:container-instance/
<container_instance_UUID>"
            ]
        }
    ]
}
```

Task Definition Examples

Task definition IAM policies do not support resource-level permissions, but the following IAM policy allows a user to register, list, and describe task definitions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RegisterTaskDefinition",
                "ecs>ListTaskDefinitions",
                "ecs:DescribeTaskDefinition"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Run Task Example

The resources for `RunTask` are task definitions. To limit which clusters a user can run task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to run any revision of a specific task definition on a specific cluster:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
{
    "Effect": "Allow",
    "Action": [
        "ecs:RunTask"
    ],
    "Condition": {
        "ArnEquals": {
            "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
        }
    },
    "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task-definition/<task_family>:*>
    ]
}
]
```

Start Task Example

The resources for `StartTask` are task definitions. To limit which clusters and container instances a user can start task definitions on, you can specify them in the `Condition` block. The advantage is that you don't have to list both task definitions and clusters in your resources to allow the appropriate access. You can apply one, the other, or both.

The following IAM policy allows permission to start any revision of a specific task definition on a specific cluster and specific container instance.

Note

For this example, when you call the `StartTask` API with the AWS CLI or another AWS SDK, you must specify the task definition revision so that the `Resource` mapping matches.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:StartTask"
            ],
            "Condition": {
                "ArnEquals": {
                    "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>",
                    "ecs:container-instances" : [
                        "arn:aws:ecs:<region>:<aws_account_id>:container-instance/<container_instance_UUID>"
                    ]
                }
            },
            "Resource": [
                "arn:aws:ecs:<region>:<aws_account_id>:task-definition/<task_family>:*>
            ]
        }
    ]
}
```

List and Describe Task Examples

The following IAM policy allows a user to list tasks for a specified cluster:

```
{
    "Version": "2012-10-17",
```

```

"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "ecs>ListTasks"
    ],
    "Condition": {
      "ArnEquals": {
        "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
      }
    },
    "Resource": [
      "*"
    ]
  }
]
}

```

The following IAM policy allows a user to describe a specified task in a specified cluster:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeTasks"
      ],
      "Condition": {
        "ArnEquals": {
          "ecs:cluster": "arn:aws:ecs:<region>:<aws_account_id>:cluster/<cluster_name>"
        }
      },
      "Resource": [
        "arn:aws:ecs:<region>:<aws_account_id>:task/<task_UUID>"
      ]
    }
  ]
}

```

Create Service Example

The following IAM policy allows a user to create Amazon ECS services in the AWS Management Console:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:Describe*",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm",
        "ecs>List*",
        "ecs:Describe*",
        "ecs>CreateService",
        "elasticloadbalancing:Describe*",
        "iam:AttachRolePolicy",
        "iam>CreateRole",
        "iam:GetPolicy",
        "iam:ListPolicies"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

```

        "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:AWSPropertyName": [
                    "ecs.amazonaws.com",
                    "spot.amazonaws.com",
                    "spotfleet.amazonaws.com",
                    "ecs.application-autoscaling.amazonaws.com",
                    "autoscaling.amazonaws.com"
                ]
            }
        }
    }
}

```

```
    ]  
}
```

AmazonEC2ContainerServiceforEC2Role

Amazon ECS attaches this policy to a service role that allows Amazon ECS to perform actions on your behalf against Amazon EC2 instances or external instances.

This policy grants administrative permissions that allow Amazon ECS container instances to make calls to AWS on your behalf. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

Considerations

You should consider the following recommendations and considerations when using the `AmazonEC2ContainerServiceforEC2Role` managed IAM policy.

- Following the standard security advice of granting least privilege, you can modify the `AmazonEC2ContainerServiceforEC2Role` managed policy to fit your specific needs. If any of the permissions granted in the managed policy aren't needed for your use case, create a custom policy and add only the permissions that you require. For example, the `UpdateContainerInstancesState` permission is provided for Spot Instance draining. If that permission isn't needed for your use case, exclude it using a custom policy. For more information, see [Permissions details \(p. 679\)](#).
- Containers that are running on your container instances have access to all of the permissions that are supplied to the container instance role through [instance metadata](#). We recommend that you limit the permissions in your container instance role to the minimal list of permissions that are provided in the managed `AmazonEC2ContainerServiceforEC2Role` policy. If the containers in your tasks need extra permissions that aren't listed, we recommend providing those tasks with their own IAM roles. For more information, see [IAM Roles for Tasks \(p. 699\)](#).

You can prevent containers on the `docker0` bridge from accessing the permissions supplied to the container instance role. You can do this while still allowing the permissions that are provided by [IAM Roles for Tasks \(p. 699\)](#) by running the following `iptables` command on your container instances. Containers can't query instance metadata with this rule in effect. This command assumes the default Docker bridge configuration and it doesn't work with containers that use the host network mode. For more information, see [Network mode \(p. 211\)](#).

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
```

You must save this `iptables` rule on your container instance for it to survive a reboot. For the Amazon ECS-optimized AMI, use the following command. For other operating systems, consult the documentation for that OS.

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo service iptables save
```

Permissions details

The `AmazonEC2ContainerServiceforEC2Role` managed IAM policy includes the following permissions. Following the standard security advice of granting least privilege, the

[AmazonEC2ContainerServiceforEC2Role](#) managed policy can be used as a guide. If you don't need any of the permissions that are granted in the managed policy for your use case, create a custom policy and add only the permissions that you need.

- `ec2:DescribeTags` – Allows a principal to describe the tags that are associated with an Amazon EC2 instance. This permission is used by the Amazon ECS container agent to support resource tag propagation. For more information, see [Tagging your resources \(p. 605\)](#).
 - `ecs>CreateCluster` – Allows a principal to create an Amazon ECS cluster. This permission is used by the Amazon ECS container agent to create a default cluster, if one doesn't already exist.
 - `ecs:DeregisterContainerInstance` – Allows a principal to deregister an Amazon ECS container instance from a cluster. The Amazon ECS container agent doesn't call this API, but this permission remains to ensure backwards compatibility.
 - `ecs:DiscoverPollEndpoint` – This action returns endpoints that the Amazon ECS container agent uses to poll for updates.
 - `ecs:Poll` – Allows the Amazon ECS container agent to communicate with the Amazon ECS control plane to report task state changes.
 - `ecs:RegisterContainerInstance` – Allows a principal to register a container instance with a cluster. This permission is used by the Amazon ECS container agent to register the Amazon EC2 instance with a cluster as well as to support resource tag propagation.
 - `ecs:StartTelemetrySession` – Allows the Amazon ECS container agent to communicate with the Amazon ECS control plane to report health information and metrics for each container and task.
 - `ecs:UpdateContainerInstancesState` – Allows a principal to modify the status of an Amazon ECS container instance. This permission is used by the Amazon ECS container agent for Spot Instance draining. For more information, see [Spot Instance Draining \(p. 367\)](#).
 - `ecs:Submit*` – This includes the `SubmitAttachmentStateChanges`, `SubmitContainerStateChange`, and `SubmitTaskStateChange` API actions. They're used by the Amazon ECS container agent to report state changes for each resource to the Amazon ECS control plane. The `SubmitContainerStateChange` permission is no longer used by the Amazon ECS container agent but remains to ensure backwards compatibility.
 - `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 is, 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 `AmazonEC2ContainerServiceforEC2Role` policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeTags",
```

```
        "ecs:CreateCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:DiscoverPollEndpoint",
        "ecs:Poll",
        "ecs:RegisterContainerInstance",
        "ecs:StartTelemetrySession",
        "ecs:UpdateContainerInstancesState",
        "ecs:Submit*",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "logs>CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "*"
}
]
}
```

AmazonEC2ContainerServiceEventsRole

This policy grants permissions that allow Amazon EventBridge (formerly CloudWatch Events) to run tasks on your behalf. This policy can be attached to the IAM role that's specified when you create scheduled tasks. For more information, see [Amazon ECS CloudWatch Events IAM Role \(p. 708\)](#).

Permissions details

This policy includes the following permissions.

- **ecs** – Allows a principal in a service to call the Amazon ECS RunTask API.
- **iam** – Allows passing any IAM service role to any Amazon ECS tasks.

The following is an example `AmazonEC2ContainerServiceEventsRole` policy.

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

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 \(p. 691\)](#).

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 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 \(p. 684\)](#).

AmazonEC2ContainerServiceRole

Important

The `AmazonEC2ContainerServiceAutoscaleRole` 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 \(p. 685\)](#).

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 \(p. 674\)](#).

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:ListHealthChecks"
            ]
        }
    ]
}
```

```

        "route53>List*",
        "route53UpdateHealthCheck",
        "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:*CreateTags"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*"
},
{
    "Sid": "CWLogGroupManagement",
    "Effect": "Allow",
    "Action": [

```

```

        "logs:CreateLogGroup",
        "logs:DescribeLogGroups",
        "logs:PutRetentionPolicy"
    ],
    "Resource": "arn:aws:logs:*:log-group:/aws/ecs/*"
},
{
    "Sid": "CWLogStreamManagement",
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:*log-group:/aws/ecs/*:log-stream:*

```

Create the service-linked role

Under most circumstances, you don't need to manually create the service-linked role. For example, when you create a new cluster (for example, with the Amazon ECS first-run experience, the cluster creation wizard, or the AWS CLI or SDKs), or create or update a service in the AWS Management Console, Amazon ECS creates the service-linked role for you, if it does not already exist.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role.

To allow an IAM entity to create the AWSServiceRoleForECS service-linked role

Add the following statement to the permissions policy for the IAM entity that needs to create the service-linked role:

```

{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "arn:aws:iam::role/aws-service-role/ecs.amazonaws.com/
AWSServiceRoleForECS*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ecs.amazonaws.com"}}
}

```

Creating a service-linked role in IAM (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to create a service-linked role with the trust policy and inline policies that the service needs to assume the role.

To create a service-linked role (CLI)

Use the following command:

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

Edit the service-linked role

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.

1. Scale all Amazon ECS services down to a desired count of 0 in all regions, and then delete the services. For more information, see [Updating a service \(p. 559\)](#) and [Deleting a service \(p. 562\)](#).
2. Force deregister all container instances from all clusters in all regions. For more information, see [Deregister an Amazon EC2 backed container instance \(p. 429\)](#).
3. Delete all Amazon ECS clusters in all regions. For more information, see [Deleting a cluster \(p. 192\)](#).

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 \(p. 283\)](#).
- 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 \(p. 693\)](#).
 - 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 \(p. 694\)](#).

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 \(p. 692\)](#). If the role does exist, select the role to view the attached policies.
4. On the **Permissions** tab, ensure that the **AmazonECSTaskExecutionRolePolicy** managed policy is attached to the role. If the policy is attached, your Amazon ECS task execution role is properly configured. If not, follow the substeps below to attach the policy.
 - a. Choose **Attach policies**.
 - b. To narrow the available policies to attach, for **Filter**, type **AmazonECSTaskExecutionRolePolicy**.
 - c. Check the box to the left of the **AmazonECSTaskExecutionRolePolicy** policy and choose **Attach policy**.
5. Choose **Trust relationships, Edit trust relationship**.
6. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Creating the task execution IAM role

If your account does not already have a task execution role, use the following steps to create the role.

To create 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 **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 **AmazonECSTaskExecutionRolePolicy**, 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**, type `ecsTaskExecutionRole` and 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 \(p. 300\)](#).

To provide access to the secrets that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see [Adding and Removing IAM Policies](#).

- `secretsmanager:GetSecretValue`
- `kms:Decrypt`—Required only if your key uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

An example inline policy adding the permissions is shown below.

```
{
```

```

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

```

Required IAM permissions for Amazon ECS secrets

To use the Amazon ECS secrets feature, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary AWS Systems Manager or Secrets Manager resources. For more information, see [Specifying sensitive data \(p. 303\)](#).

To provide access to the AWS Systems Manager Parameter Store parameters that you create, manually add the following permissions as an inline policy to the task execution role. For more information, see [Adding and Removing IAM Policies](#).

- `ssm:GetParameters`—Required if you are referencing a Systems Manager Parameter Store parameter in a task definition.
- `secretsmanager:GetSecretValue`—Required if you are referencing a Secrets Manager secret either directly or if your Systems Manager Parameter Store parameter is referencing a Secrets Manager secret in a task definition.
- `kms:Decrypt`—Required only if your secret uses a custom KMS key and not the default key. The ARN for your custom key should be added as a resource.

The following example inline policy adds the required permissions:

```

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

```

Optional IAM permissions for Fargate tasks pulling Amazon ECR images over interface endpoints

When launching tasks that use the Fargate launch type that pull images from Amazon ECR when Amazon ECR is configured to use an interface VPC endpoint, you can restrict the tasks access to a specific

VPC or VPC endpoint. Do this by creating a task execution role for the tasks to use that use IAM condition keys.

Use the following IAM global condition keys to restrict access to a specific VPC or VPC endpoint. For more information, see [AWS Global Condition Context Keys](#).

- `aws:SourceVpc`—Restricts access to a specific VPC.
- `aws:SourceVpce`—Restricts access to a specific VPC endpoint.

The following task execution role policy provides an example for adding condition keys:

Important

The `ecr:GetAuthorizationToken` API action cannot have the `aws:sourceVpc` or `aws:sourceVpce` condition keys applied to it because the `GetAuthorizationToken` API call goes through the elastic network interface owned by AWS Fargate rather than the elastic network interface of the task.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ecr:GetAuthorizationToken",  
                "logs>CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ecr:BatchCheckLayerAvailability",  
                "ecr:GetDownloadUrlForLayer",  
                "ecr:BatchGetImage"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:sourceVpce": "vpce-xxxxxx",  
                    "aws:sourceVpc": "vpc-xxxxx"  
                }  
            }  
        }  
    ]  
}
```

Amazon ECS container instance IAM role

Amazon ECS container instances, including both Amazon EC2 and external instances, run the Amazon ECS container agent and require an IAM role for the service to know that the agent belongs to you. Before you launch container instances and register them to a cluster, you must create an IAM role for your container instances to use.

Important

If you are registering external instances to your cluster, the IAM role you use requires Systems Manager permissions as well. For more information, see [Required IAM permissions for external instances \(p. 416\)](#).

Amazon ECS provides the `AmazonEC2ContainerServiceforEC2Role` managed IAM policy which contains the permissions needed to use the full Amazon ECS feature set. This managed

policy can be attached to an IAM role and associated with your container instances. Alternatively, you can use the managed policy as a guide when creating a custom policy to use. The container instance role provides permissions needed for the Amazon ECS container agent and Docker daemon to call AWS APIs on your behalf. For more information on the managed policy, see [AmazonEC2ContainerServiceforEC2Role \(p. 679\)](#).

Creating the Amazon ECS container instance IAM role

Important

If you are registering external instances to your cluster, see [Required IAM permissions for external instances \(p. 416\)](#).

The Amazon ECS instance role is automatically created for you when completing the Amazon ECS console first-run experience. However, you can manually create the role and attach the managed IAM policy for container instances to allow Amazon ECS to add permissions for future features and enhancements as they are introduced. Use the following procedure to check and see if your account already has the Amazon ECS container instance IAM role and to attach the managed IAM policy if needed.

To check for the `ecsInstanceRole` in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsInstanceRole`. If the role does not exist, use the procedure in the next section to create the role. If the role does exist, select the role to view the attached policies.
4. Choose the **Permissions** tab.
5. In the **Permissions policies** section, ensure that the **AmazonEC2ContainerServiceforEC2Role** managed policy, or an equivalent custom policy, is attached to the role. If the policy is attached, your container instance role is properly configured. If not, follow the substeps below to attach the policy.

Important

The **AmazonEC2ContainerServiceforEC2Role** managed policy should be attached to the container instance IAM role, otherwise you will receive an error using the AWS Management Console to create clusters.

- a. Choose **Attach policies**.
- b. In the **Filter policies** search box, type **AmazonEC2ContainerServiceforEC2Role** to narrow the available policies to attach.
- c. Check the box to the left of the **AmazonEC2ContainerServiceforEC2Role** policy and choose **Attach policy**.
6. Choose the **Trust relationships** tab, and **Edit trust relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "ec2.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

}

To create the `ecsInstanceRole` IAM role for your container instances

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. Choose the **AWS service** role type, and then choose **Elastic Container Service**.
4. Choose the **EC2 Role for Elastic Container Service** use case and then **Next: Permissions**.
5. In the **Attached permissions policies** section, verify the **AmazonEC2ContainerServiceforEC2Role** policy is selected and then choose **Next:Tags**.

Important

The **AmazonEC2ContainerServiceforEC2Role** managed policy should be attached to the container instance IAM role, otherwise you will receive an error using the AWS Management Console to create clusters.

6. For **Add tags (optional)**, specify any custom tags to associate with the policy and then choose **Next: Review**.
7. For **Role name**, type `ecsInstanceRole` and optionally you can enter a description.
8. Review your role information and then choose **Create role** to finish.

Adding Amazon S3 read-only access to your container instance IAM role

Storing configuration information in a private bucket in Amazon S3 and granting read-only access to your container instance IAM role is a secure and convenient way to allow container instance configuration at launch time. You can store a copy of your `ecs.config` file in a private bucket, use Amazon EC2 user data to install the AWS CLI and then copy your configuration information to `/etc/ecs/ecs.config` when the instance launches.

For more information about creating an `ecs.config` file, storing it in Amazon S3, and launching instances with this configuration, see [Storing Container Instance Configuration in Amazon S3 \(p. 467\)](#).

To allow Amazon S3 read-only access for your container instance role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Choose the IAM role you use for your container instances (this role is likely titled `ecsInstanceRole`). For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
4. Choose the **Permissions** tab, then **Attach policy**.
5. On the **Attach policy** page, type `S3` into the **Filter: Policy type** field to narrow the policy results.
6. Check the box to the left of the **AmazonS3ReadOnlyAccess** policy and click **Attach policy**.

Note

This policy allows read-only access to all Amazon S3 resources. For more restrictive bucket policy examples, see [Bucket Policy Examples](#) in the Amazon Simple Storage Service User Guide.

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
```

IAM Roles for Tasks

With IAM roles for Amazon ECS tasks, you can specify an IAM role that can be used by the containers in a task. Applications must sign their AWS API requests with AWS credentials, and this feature provides a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the EC2 instance's role, you can associate an IAM role with an ECS task definition or `RunTask` API operation. The applications in the task's containers can then use the AWS SDK or CLI to make API requests to authorized AWS services.

Important

Containers that are running on your container instances are not prevented from accessing the credentials that are supplied to the container instance profile (through the Amazon EC2 instance metadata server). We recommend that you limit the permissions in your container instance role to the minimal list of permissions shown in [Amazon ECS container instance IAM role \(p. 695\)](#).

To prevent containers in tasks that use the awsVpc network mode from accessing the credential information supplied to the container instance profile (while still allowing the permissions that are provided by the task role), set the `ECS_AWSVPC_BLOCK_IMDS` agent configuration variable to true in the agent configuration file and restart the agent. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

To prevent containers in tasks that use the bridge network mode from accessing the credential information supplied to the container instance profile (while still allowing the permissions that are provided by the task role) by running the following `iptables` command on your container instances. Note that this command does not affect containers in tasks that use the host or awsVpc network modes. For more information, see [Network mode \(p. 211\)](#).

```
sudo yum install -y iptables-services; sudo iptables --insert FORWARD 1 --in-interface docker+ --destination 169.254.169.254/32 --jump DROP
```

You must save this `iptables` rule on your container instance for it to survive a reboot. For the Amazon ECS-optimized AMI, use the following command. For other operating systems, consult the documentation for that OS.

- For the Amazon ECS-optimized Amazon Linux 2 AMI:

```
sudo iptables-save | sudo tee /etc/sysconfig/iptables && sudo systemctl enable --now iptables
```

- For the Amazon ECS-optimized Amazon Linux AMI:

```
sudo service iptables save
```

You define the IAM role to use in your task definitions, or you can use a `taskRoleArn` override when running a task manually with the `RunTask` API operation. The Amazon ECS agent receives a payload message for starting the task with additional fields that contain the role credentials. The Amazon ECS agent sets a unique task credential ID as an identification token and updates its internal credential cache so that the identification token for the task points to the role credentials that are received in the payload. The Amazon ECS agent populates the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable in the `Env` object (available with the `docker inspect container_id` command) for all containers that belong to this task with the following relative URI: `/credential_provider_version/credentials?id=task_credential_id`.

Note

When you specify an IAM role for a task, the AWS CLI or other SDKs in the containers for that task use the AWS credentials provided by the task role exclusively and they no longer inherit any IAM permissions from the container instance.

From inside the container, you can query the 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"
}
```

Note

The default expiration time for the generated IAM role credentials is 6 hours.

If your container instance is using at least version 1.11.0 of the container agent and a supported version of the AWS CLI or SDKs, then the SDK client will see that the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` variable is available, and it will use the provided credentials to make calls to the AWS APIs. For more information, see [Enabling Task IAM Roles on your Container Instances \(p. 701\)](#) and [Using a Supported AWS SDK \(p. 704\)](#).

Each time the credential provider is used, the request is logged locally on the host container instance at `/var/log/ecs/audit.log.YYYY-MM-DD-HH`. For more information, see [IAM Roles for Tasks Credential Audit Log \(p. 831\)](#).

Topics

- [Benefits of Using IAM Roles for Tasks \(p. 701\)](#)
- [Enabling Task IAM Roles on your Container Instances \(p. 701\)](#)
- [Creating an IAM Role and Policy for your Tasks \(p. 702\)](#)
- [Using a Supported AWS SDK \(p. 704\)](#)
- [Specifying an IAM Role for your Tasks \(p. 704\)](#)

Benefits of Using IAM Roles for Tasks

- **Credential Isolation:** A container can only retrieve credentials for the IAM role that is defined in the task definition to which it belongs; a container never has access to credentials that are intended for another container that belongs to another task.
- **Authorization:** Unauthorized containers cannot access IAM role credentials defined for other tasks.
- **Auditability:** Access and event logging is available through CloudTrail to ensure retrospective auditing. Task credentials have a context of `taskArn` that is attached to the session, so CloudTrail logs show which task is using which role.

Enabling Task IAM Roles on your Container Instances

Your Amazon ECS container instances require at least version 1.11.0 of the container agent to enable task IAM roles; however, we recommend using the latest container agent version. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#). If you are using the Amazon ECS-optimized AMI, your instance needs at least 1.11.0-1 of the `ecs-init` package. If your container instances are launched from version 2016.03.e or later, then they contain the required versions of the container agent and `ecs-init`. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).

If you are not using the Amazon ECS-optimized AMI for your container instances, be sure to add the `--net=host` option to your `docker run` command that starts the agent and the appropriate agent configuration variables for your desired configuration (for more information, see [Amazon ECS container agent configuration \(p. 454\)](#)):

```
ECS_ENABLE_TASK_IAM_ROLE=true
```

Enables IAM roles for tasks for containers with the `bridge` and `default` network modes.

ECS_ENABLE_TASK_IAM_ROLE_NETWORK_HOST=true

Enables IAM roles for tasks for containers with the host network mode. This variable is only supported on agent versions 1.12.0 and later.

For an example run command, see [Manually updating the Amazon ECS container agent \(for non-Amazon ECS-Optimized AMIs\) \(p. 452\)](#). You will also need to set the following networking commands on your container instance so that the containers in your tasks can retrieve their AWS credentials:

```
sudo sysctl -w net.ipv4.conf.all.route_localnet=1
sudo iptables -t nat -A PREROUTING -p tcp -d 169.254.170.2 --dport 80 -j DNAT --to-
destination 127.0.0.1:51679
sudo iptables -t nat -A OUTPUT -d 169.254.170.2 -p tcp -m tcp --dport 80 -j REDIRECT --to-
ports 51679
```

You must save these **iptables** rules on your container instance for them to survive a reboot. You can use the **iptables-save** and **iptables-restore** commands to save your **iptables** rules and restore them at boot. For more information, consult your specific operating system documentation.

Creating an IAM Role and Policy for your Tasks

You must create an IAM policy for your tasks to use that specifies the permissions that you would like the containers in your tasks to have. You have several ways to create a new IAM permission policy. You can copy a complete AWS managed policy that already does some of what you're looking for and then customize it to your specific requirements. For more information, see [Creating a New Policy](#) in the *IAM User Guide*.

You must also create a role for your tasks to use before you can specify it in your task definitions. You can create the role using the **Amazon Elastic Container Service Task Role** service role in the IAM console. Then you can attach your specific IAM policy to the role that gives the containers in your task the permissions you desire. The procedures below describe how to do this.

If you have multiple task definitions or services that require IAM permissions, you should consider creating a role for each specific task definition or service with the minimum required permissions for the tasks to operate so that you can minimize the access that you provide for each task.

For information about the service endpoint for your Region, see [Service endpoints in the Amazon Web Services General Reference Reference Guide](#).

The Amazon ECS Task Role trust relationship is shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To create an IAM policy for your tasks

In this example, we create a policy to allow read-only access to an Amazon S3 bucket. You could store database credentials or other secrets in this bucket, and the containers in your task can read the credentials from the bucket and load them into your application.

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Follow the steps under one of the following tabs, which shows you how to use the visual or JSON editors.

Using the visual editor

1. For **Service**, choose **S3**.
2. For **Actions**, expand the **Read** option and select **GetObject**.
3. For **Resources**, select **Add ARN** and enter the full Amazon Resource Name (ARN) of your Amazon S3 bucket, and then choose **Review policy**.
4. On the **Review policy** page, for **Name** type your own unique name, such as **AmazonECSTaskS3BucketPolicy**.
5. Choose **Create policy** to finish.

Using the JSON editor

1. In the **Policy Document** field, paste the policy to apply to your tasks. The example below allows permission to the **my-task-secrets-bucket** Amazon S3 bucket. You can modify the policy document to suit your specific needs.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::my-task-secrets-bucket/*"  
            ]  
        }  
    ]  
}
```

2. Choose **Create policy**.

To create an IAM role for your tasks

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**, **Create role**.
3. For **Select type of trusted entity** section, choose **AWS service**.
4. For **Choose the service that will use this role**, choose **Elastic Container Service**.
5. For **Select your use case**, choose **Elastic Container Service Task** and choose **Next: Permissions**.
6. For **Attach permissions policy**, select the policy to use for your tasks (in this example **AmazonECSTaskS3BucketPolicy**), and then choose **Next: Tags**.
7. For **Add tags (optional)**, enter any metadata tags you want to associate with the IAM role, and then choose **Next: Review**.
8. For **Role name**, enter a name for your role. For this example, type **AmazonECSTaskS3BucketRole** to name the role, and then choose **Create role** to finish.

Using a Supported AWS SDK

Support for IAM roles for tasks was added to the AWS SDKs on July 13th, 2016. The containers in your tasks must use an AWS SDK version that was created on or after that date. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature.

To ensure that you are using a supported SDK, follow the installation instructions for your preferred SDK at [Tools for Amazon Web Services](#) when you are building your containers to get the latest version.

Specifying an IAM Role for your Tasks

After you have created a role and attached a policy to that role, you can run tasks that assume the role. You have several options to do this:

- Specify an IAM role for your tasks in the task definition. You can create a new task definition or a new revision of an existing task definition and specify the role you created previously. If you use the console to create your task definition, choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the `taskRoleArn` parameter. For more information, see [Creating a task definition using the new console \(p. 196\)](#).

Note

This option is required if you want to use IAM task roles in an Amazon ECS service.

- Specify an IAM task role override when running a task. You can specify an IAM task role override when running a task. If you use the console to run your task, choose **Advanced Options** and then choose your IAM role in the **Task Role** field. If you use the AWS CLI or SDKs, specify your task role ARN using the `taskRoleArn` parameter in the `overrides` JSON object. For more information, see [Run a standalone task \(p. 510\)](#).

Note

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

Note IAM users also require permissions to use CodeDeploy; these permissions are described in [Blue/green deployment required IAM permissions \(p. 568\)](#).

There are two managed policies provided. The `AWSCodeDeployRoleForECS` policy, shown below, gives CodeDeploy permission to update any resource using the associated action.

```

        "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: Permissions**.
6. Choose **Next: Tags**.
7. For **Add tags (optional)**, you can add optional IAM tags to the role. Choose **Next:Review** when finished.
8. For **Role name**, type `ecsCodeDeployRole`, enter an optional description, and then choose **Create role**.

To add the required permissions to the Amazon ECS CodeDeploy IAM role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Search the list of roles for `ecsCodeDeployRole`. If the role does not exist, use the procedure above to create the role. If the role does exist, select the role to view the attached policies.
3. In the **Permissions policies** section, ensure that either the **AWSCodeDeployRoleForECS** or **AWSCodeDeployRoleForECSLimited** managed policy is attached to the role. If the policy is

attached, your Amazon ECS CodeDeploy service role is properly configured. If not, follow the substeps below to attach the policy.

- a. Choose **Attach policies**.
- b. To narrow the available policies to attach, for **Filter**, type **AWSCodeDeployRoleForECS** or **AWSCodeDeployRoleForECSLimited**.
- c. Check the box to the left of the AWS managed policy and choose **Attach policy**.
4. Choose **Trust Relationships**, **Edit trust relationship**.
5. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "codedeploy.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

6. If the tasks in your Amazon ECS service using the blue/green deployment type require the use of the task execution role or a task role override, then you must add the `iam:PassRole` permission for each task execution role or task role override to the CodeDeploy IAM role as an inline policy. For more information, see [Amazon ECS task execution IAM role \(p. 691\)](#) and [IAM Roles for Tasks \(p. 699\)](#).

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 \(p. 524\)](#).

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 \(p. 691\)](#).

Note

Specify the full ARN of your task execution role or task role override.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:PassRole",  
            "Resource": [  
                "arn:aws:iam::<aws_account_id>:role/  
                <ecsTaskExecutionRole_or_TaskRole_name>"  
            ]  
        }  
    ]  
}
```

```
    ]  
}
```

You can use the following procedure to check that your account already has the CloudWatch Events IAM role, and manually create it if needed.

To check for the CloudWatch Events IAM role in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `ecsEventsRole`. If the role does not exist, use the next procedure to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**.
5. In the **Permissions policies** section, ensure that the **AmazonEC2ContainerServiceEventsRole** managed policy is attached to the role. If the policy is attached, your Amazon ECS service role is properly configured. If not, follow the substeps below to attach the policy.
 - a. Choose **Attach policies**.
 - b. To narrow the available policies to attach, for **Filter**, type `AmazonEC2ContainerServiceEventsRole`.
 - c. Select the box to the left of the **AmazonEC2ContainerServiceEventsRole** policy and choose **Attach policy**.
6. Choose **Trust relationships, Edit trust relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "",  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "events.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```

To create an IAM role for CloudWatch Events

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles** and then choose **Create role**.
3. In the **Select type of trusted entity** section, choose **Elastic Container Service**. For **Select your use case** choose **Elastic Container Service Task**. Choose **Next: Permissions**.
4. In the **Attach permissions policy** section, select the **AmazonEC2ContainerServiceEventsRole** policy and choose **Next: Tags**.
5. In the **Add tags (optional)** section, enter any tags you would like to associate with the role and choose **Next: Review**.
6. For **Role name**, type `ecsEventsRole` to name the role, optionally enter a description, and then choose **Create role**.
7. Review your role information and choose **Create Role**.

8. Search the list of roles for `ecsEventsRole` and select the role you just created.
9. Choose **Trust relationships**, **Edit trust relationship**.
10. Replace the existing trust relationship with the following text in the **Policy Document** window and choose **Update Trust Policy**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "events.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

To add permissions for the task execution role to the CloudWatch Events IAM role

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**, **Create policy**.
3. Choose **JSON**, paste the following policy, and then choose **Review policy**:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "iam:PassRole",  
            "Resource": [  
                "arn:aws:iam:<aws_account_id>:role/  
                <ecsTaskExecutionRole_or_TaskRole_name>"  
            ]  
        }  
    ]  
}
```

4. For **Name**, type `AmazonECSEventsTaskExecutionRole`, optionally enter a description, and then choose **Create policy**.
5. In the navigation pane, choose **Roles**.
6. Search the list of roles for `ecsEventsRole` and select the role to view the attached policies.
7. Choose **Attach policy**.
8. In the **Attach policy** section, select the `AmazonECSEventsTaskExecutionRole` policy and choose **Attach policy**.

Troubleshooting Amazon Elastic Container Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon ECS and IAM.

Topics

- [I'm not authorized to perform an action in Amazon ECS \(p. 711\)](#)
- [I'm not authorized to perform iam:PassRole \(p. 711\)](#)
- [I want to view my access keys \(p. 711\)](#)
- [I'm an administrator and want to allow others to access Amazon ECS \(p. 712\)](#)
- [I want to allow people outside of my AWS account to access my Amazon ECS resources \(p. 712\)](#)

I'm not authorized to perform an action in Amazon ECS

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a `widget` but does not have `ecs:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
  ecs:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-example-widget` resource using the `ecs:GetWidget` action.

I'm 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 \(p. 658\)](#).
- 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 \(p. 620\)](#).

For clusters with tasks or services using the EC2 launch type, you can use CloudWatch alarms to scale in and scale out the container instances based on CloudWatch metrics, such as cluster memory reservation. For more information, see [Tutorial: Scaling container instances with CloudWatch alarms \(p. 628\)](#).

Amazon CloudWatch Logs

Monitor, store, and access the log files from the containers in your Amazon ECS tasks by specifying the `awslogs` log driver in your task definitions. This is the only supported method for accessing logs for tasks using the Fargate launch type, but also works with tasks using the EC2 launch type. For more information, see [Using the awslogs log driver \(p. 283\)](#).

You can also monitor, store, and access the operating system and Amazon ECS container agent log files from your Amazon ECS container instances. This method for accessing logs can be used for containers using the EC2 launch type. For more information, see [Monitoring your container instances \(p. 426\)](#).

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 \(p. 632\)](#) 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 \(p. 650\)](#).

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

For more information, see [AWS Trusted Advisor](#) in the *AWS Support User Guide*.

Another important part of monitoring Amazon ECS involves manually monitoring those items that the CloudWatch alarms don't cover. The CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your container instances and the containers in your tasks.

Compliance Validation for Amazon Elastic Container Service

Third-party auditors assess the security and compliance of Amazon Elastic Container Service as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading reports in AWS Artifact](#).

Your compliance responsibility when using Amazon ECS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [AWS Config](#) – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Infrastructure Security in Amazon Elastic Container Service

As a managed service, Amazon Elastic Container Service is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access Amazon ECS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location, but Amazon ECS does support resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon ECS policies to control access from specific Amazon Virtual Private Cloud endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon ECS resource from only the specific VPC within the AWS network. For more information, see [Amazon ECS interface VPC endpoints \(AWS PrivateLink\)](#) (p. 714).

Topics

- [Amazon ECS interface VPC endpoints \(AWS PrivateLink\)](#) (p. 714)

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 \(p. 694\)](#).

- To allow your tasks to pull sensitive data from Secrets Manager, you must create the interface VPC endpoints for Secrets Manager. For more information, see [Using Secrets Manager with VPC Endpoints](#) in the *AWS Secrets Manager User Guide*.
- If your VPC doesn't have an internet gateway and your tasks use the awslogs log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see [Using CloudWatch Logs with Interface VPC Endpoints](#) in the *Amazon CloudWatch Logs User Guide*.
- Tasks using the EC2 launch type require that the container instances that they're launched on to run version 1.25.1 or later of the Amazon ECS container agent. For more information, see [Amazon ECS container agent versions \(p. 440\)](#).
- 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.

If you have existing tasks that are using the EC2 launch type, after you have created the VPC endpoints, each container instance needs to pick up the new configuration. For this to happen, you must either reboot each container instance or restart the Amazon ECS container agent on each container instance. To restart the container agent, do the following.

To restart the Amazon ECS container agent

1. Log in to your container instance via SSH. For more information, see [Connect to your container instance \(p. 384\)](#).
2. Stop the container agent.

```
sudo docker stop ecs-agent
```

3. Start the container agent.

```
sudo docker start ecs-agent
```

After you have created the VPC endpoints and restarted the Amazon ECS container agent on each container instance, all newly launched tasks pick up the new configuration.

Create the Secrets Manager and Systems Manager endpoints

If you are referencing either Secrets Manager secrets or Systems Manager Parameter Store parameters in your task definitions to inject sensitive data into your containers, you need to create the interface VPC endpoints for Secrets Manager or Systems Manager so those tasks can reach those services. You only need to create the endpoints from the specific service your sensitive data is hosted in. For more information, see [Specifying sensitive data \(p. 303\)](#).

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 endpoints

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 \(p. 805\)](#).

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 the listed Amazon ECS actions for all principals on all resources.

```
{
```

```
"Statement": [
    {
        "Principal": "*",
        "Effect": "Allow",
        "Action": [
            "ecs:action-1",
            "ecs:action-2",
            "ecs:action-2"
        ],
        "Resource": "*"
    }
]
```

Common use cases in Amazon ECS

This topic provides guidance for two common use cases in Amazon ECS: microservices and batch jobs. Here you can find considerations and external resources that may be useful for getting your application running on Amazon ECS, and the common aspects of each solution.

Topics

- [Microservices \(p. 718\)](#)
- [Batch jobs \(p. 720\)](#)

Microservices

Microservices are built with a software architectural method that decomposes complex applications into smaller, independent services. Containers are optimal for running small, decoupled services, and they offer the following advantages:

- Containers make services easy to model in an immutable image with all of your dependencies.
- Containers can use any application and any programming language.
- The container image is a versioned artifact, so you can track your container images to the source they came from.
- You can test your containers locally, and deploy the same artifact to scale.

The following sections cover some of the aspects and challenges that you must consider when designing a microservices architecture to run on Amazon ECS. You can also view the microservices reference architecture on GitHub. For more information, see [Deploying Microservices with Amazon ECS, AWS CloudFormation, and an Application Load Balancer](#).

Topics

- [Auto Scaling \(p. 718\)](#)
- [Service discovery \(p. 719\)](#)
- [Authorization and secrets management \(p. 719\)](#)
- [Logging \(p. 719\)](#)
- [Continuous integration and continuous deployment \(p. 719\)](#)

Auto Scaling

The application load for your microservice architecture can change over time. A responsive application can scale out or in, depending on actual or anticipated load. Amazon ECS provides you with several tools to scale not only your services that are running in your clusters, but the actual clusters themselves.

For example, Amazon ECS provides CloudWatch metrics for your clusters and services. For more information, see [Amazon ECS CloudWatch metrics \(p. 620\)](#). You can monitor the memory and CPU utilization for your clusters and services. Then, use those metrics to trigger CloudWatch alarms that can automatically scale out your cluster when its resources are running low. Scale them back in when you don't need as many resources. For more information, see [Tutorial: Scaling container instances with CloudWatch alarms \(p. 628\)](#).

In addition to scaling your cluster size, your Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust its desired count up or down in response to CloudWatch alarms. Service Auto Scaling is available in all regions that support Amazon ECS. For more information, see [Service auto scaling \(p. 591\)](#).

Service discovery

Service discovery is a key component of most distributed systems and service-oriented architectures. With service discovery, your microservice components are automatically discovered as they get created and terminated on a given infrastructure. There are several approaches that you can use to make your services discoverable. The following resources describe a few examples:

- [Run Containerized Microservices with Amazon EC2 Container Service and Application Load Balancer](#): This post describes how to use the dynamic port mapping and path-based routing features of Elastic Load Balancing Application Load Balancers to provide service discovery for a microservice architecture.
- [Amazon Elastic Container Service - Reference Architecture: Service Discovery](#): This Amazon ECS reference architecture provides service discovery to containers using CloudWatch Events, Lambda, and Route 53 private hosted zones.
- [Service Discovery via Consul with Amazon ECS](#): This post shows how a third party tool called [Consul by HashiCorp](#) can augment the capabilities of Amazon ECS by providing service discovery for an ECS cluster (complete with an example application).

Authorization and secrets management

Managing secrets, such as database credentials for an application, has always been a challenging issue. The [Managing Secrets for Amazon ECS Applications Using Parameter Store and IAM Roles for Tasks](#) post focuses on how to integrate the [IAM roles for tasks \(p. 699\)](#) functionality of Amazon ECS with the AWS Systems Manager Parameter Store. Parameter Store provides a centralized store to manage your configuration data, whether it's plaintext data such as database strings or secrets such as passwords, encrypted through AWS Key Management Service.

Logging

You can configure your container instances to send log information to CloudWatch Logs. This enables you to view different logs from your container instances in one convenient location. For more information about getting started using CloudWatch Logs on your container instances that were launched with the Amazon ECS-optimized AMI, see [Monitoring your container instances \(p. 426\)](#).

You can configure the containers in your tasks to send log information to CloudWatch Logs. This enables you to view different logs from your containers in one convenient location, and it prevents your container logs from taking up disk space on your container instances. For more information about getting started using the `awslogs` log driver in your task definitions, see [Using the awslogs log driver \(p. 283\)](#).

Continuous integration and continuous deployment

Continuous integration and continuous deployment (CICD) is a common process for microservice architectures that are based on Docker containers. You can create a pipeline that takes the following actions:

- Monitors changes to a source code repository
- Builds a new Docker image from that source
- Pushes the image to an image repository such as Amazon ECR or Docker Hub
- Updates your Amazon ECS services to use the new image in your application

The following resources outline how to do this in different ways:

- [ECS Reference Architecture: Continuous Deployment](#): This reference architecture demonstrates how to achieve continuous deployment of an application to Amazon ECS using CodePipeline, CodeBuild, and AWS CloudFormation.
- [Continuous Delivery Pipeline for Amazon ECS Using Jenkins, GitHub, and Amazon ECR](#): This AWS labs repository helps you set up and configure a continuous delivery pipeline for Amazon ECS using Jenkins, GitHub, and Amazon ECR.

Batch jobs

Docker containers are particularly suited for batch job workloads. Batch jobs are often short-lived and embarrassingly parallel. You can package your batch processing application into a Docker image so that you can deploy it anywhere, such as in an Amazon ECS task. For fully managed batch processing at any scale, you should consider using AWS Batch. AWS Batch enables developers, scientists, and engineers to easily and efficiently run hundreds of thousands of batch computing jobs on AWS. AWS Batch dynamically provisions the optimal quantity and type of compute resources (for example, CPU or memory-optimized instances) based on the volume and specific resource requirements of the batch jobs submitted. For more information, see [the AWS Batch product detail pages](#).

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 \(p. 721\)](#)
- [Creating Amazon ECS resources with AWS CloudFormation \(p. 722\)](#)
- [Amazon Elastic Container Service on AWS Outposts \(p. 722\)](#)
- [Use App Mesh with Amazon ECS \(p. 726\)](#)
- [AWS Deep Learning Containers on Amazon ECS \(p. 726\)](#)

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](#)

Amazon Elastic Container Service on AWS Outposts

AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that

you use in the AWS Cloud. Amazon ECS on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see the [AWS Outposts User Guide](#).

Prerequisites

The following are the prerequisites for using Amazon ECS on AWS Outposts:

- You must have installed and configured an AWS Outposts in your on-premises data center.
- You must have a reliable network connection between your AWS Outposts and its AWS Region.
- You must have sufficient capacity of instance types available in your AWS Outposts.
- All Amazon ECS container instances must have Amazon ECS container agent 1.33.0 or later.

Limitations

The following are the limitations of using Amazon ECS on AWS Outposts:

- Amazon Elastic Container Registry, AWS Identity and Access Management, Network Load Balancer, Classic Load Balancer, and Amazon Route 53 run in the AWS Region, not on AWS Outposts. This will increase latencies between these services and the containers.
- AWS Fargate is not available on AWS Outposts.

Network Connectivity Considerations

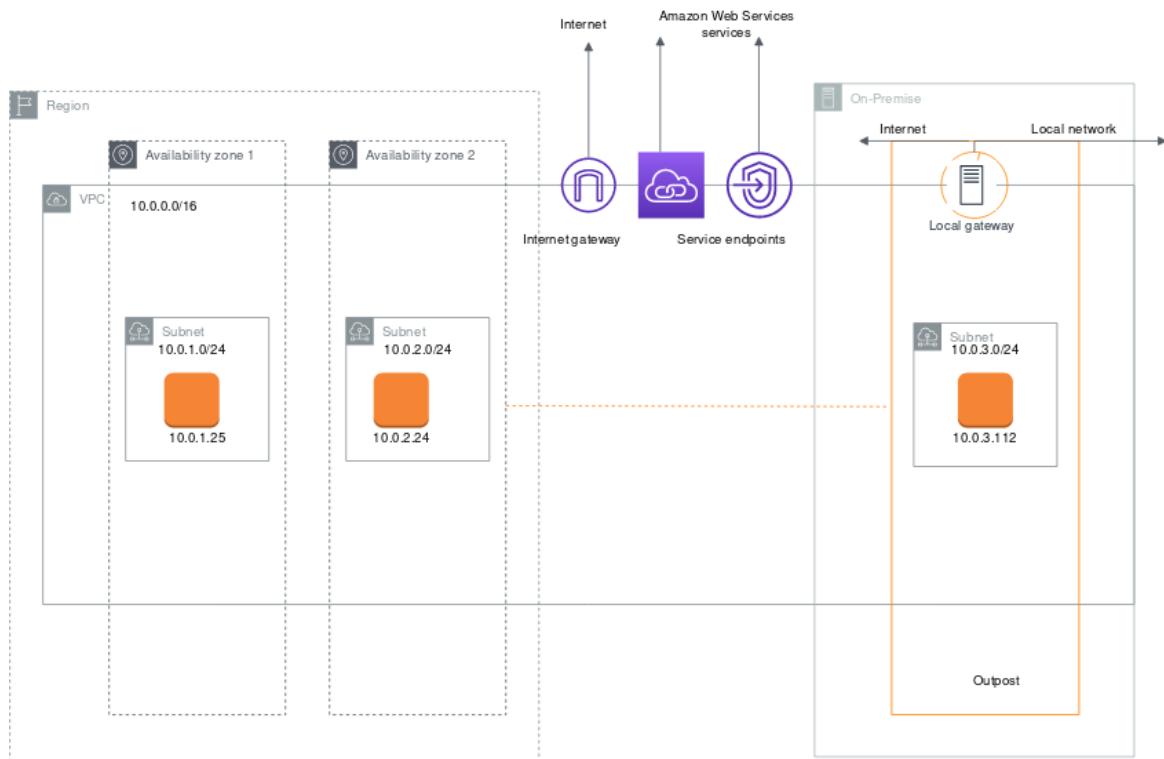
The following are network connectivity considerations for AWS Outposts:

- If network connectivity between your AWS Outposts and its AWS Region is lost, your clusters will continue to run. However, you cannot create new clusters or take new actions on existing clusters until connectivity is restored. In case of instance failures, the instance will not be automatically replaced. The CloudWatch Logs agent will be unable to update logs and event data.
- We recommend that you provide reliable, highly available, and low latency connectivity between your AWS Outposts and its AWS Region.

Creating an Amazon ECS Cluster on an AWS Outposts

Creating an Amazon ECS cluster on an AWS Outposts is similar to creating an Amazon ECS cluster in the AWS Cloud. When you create an Amazon ECS cluster on an AWS Outposts, you must specify a subnet associated with your AWS Outposts.

An AWS Outposts is an extension of an AWS Region, and you can extend an Amazon VPC in an account to span multiple Availability Zones and any associated AWS Outposts. When you configure your AWS Outposts, you associate a subnet with it to extend your Regional VPC environment to your on-premises facility. Instances on an AWS Outposts appear as part of your Regional VPC, similar to an Availability Zone with associated subnets.



AWS CLI

To create an Amazon ECS cluster on an AWS Outposts with the AWS CLI, specify a security group and a subnet that is associated with your AWS Outposts.

To create a subnet associated with your AWS Outposts.

```
aws ec2 create-subnet \
--cidr-block 10.0.3.0/24 \
--vpc-id vpc-xxxxxxxx \
--outpost-arn arn:aws:outposts:us-west-2:123456789012:outpost/op-xxxxxxxxxxxxxx \
--availability-zone-id usw2-az1
```

The following example creates an Amazon ECS cluster on an AWS Outposts.

1. Create a role and policy with rights on AWS Outposts.

```
aws iam create-role --role-name ecsRole \
-assume-role-policy-document file://ecs-policy.json
aws iam put-role-policy --role-name ecsRole --policy-name ecsRolePolicy \
--policy-document file://role-policy.json
```

2. Create an IAM instance profile with rights on AWS Outposts.

```
aws iam create-instance-profile --instance-profile-name outpost
aws iam add-role-to-instance-profile --instance-profile-name outpost \
--role-name ecsRole
```

3. Create a VPC.

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16
```

4. Create a security group for the container instances, specifying the proper CIDR range for the AWS Outposts. (This step is different for AWS Outposts.)

```
aws ec2 create-security-group --group-name MyOutpostSG
aws ec2 authorize-security-group-ingress --group-name MyOutpostSG --protocol tcp \
--port 22 --cidr 10.0.3.0/24
aws ec2 authorize-security-group-ingress ----group-name MyOutpostSG ----protocol tcp \
--port 80 --cidr 10.0.3.0/24
```

5. Create the Cluster.
6. Define the Amazon ECS container agent environment variables to launch the instance into the cluster created in the previous step and define any necessary tags.

```
#! /bin/bash
cat << 'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=MyCluster
ECS_IMAGE_PULL_BEHAVIOR=prefer-cached
ECS_CONTAINER_INSTANCE_TAGS={"environment": "Outpost"}
EOF
```

Note

In order to avoid delays caused by pulling container images from Amazon ECR in the Region, use image caches. To do this, each time a task is run, configure the Amazon ECS agent to default to using the cached image on the instance itself by setting `ECS_IMAGE_PULL_BEHAVIOR` to `prefer-cached`.

7. Create the container instance, specifying the VPC and subnet for the AWS Outposts where this instance should run and an instance type that is available on the AWS Outposts. (This step is different for AWS Outposts.)

```
aws ec2 run-instances --count 1 --image-id ami-xxxxxxxxxx --instance-type c5.large \
--key-name aws-outpost-key --subnet-id subnet-xxxxxxxxxxxxxxxxxx \
--iam-instance-profile Name outpost --security-group-id sg-xxxxxx \
--associate-public-ip-address --user-data file://userdata.txt
```

Note

This command is also used when adding additional instances to the cluster. Any containers deployed in the cluster will be placed on that specific AWS Outposts.

8. Register a task definition.

```
aws ecs register-task-definition ----cli-input-json file://ecs-task.json
```

9. Run the task or create the service.

Run the task

```
aws ecs run-task --cluster mycluster --count 1 --task-definition outpost-app:1
```

Create the service

```
aws ecs create-service --cluster mycluster --service-name outpost-service \
--task-definition outpost-app:1 --desired-count 1
```

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.

AWS Deep Learning Containers on Amazon ECS

AWS Deep Learning Containers provide a set of Docker images for training and serving models in TensorFlow and Apache MXNet (Incubating) on Amazon ECS. Deep Learning Containers enable optimized environments with TensorFlow, NVIDIA CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries. Container images for Deep Learning Containers are available in Amazon ECR to reference in Amazon ECS task definitions. You can use Deep Learning Containers along with Amazon Elastic Inference to lower your inference costs.

To get started using Deep Learning Containers without Elastic Inference on Amazon ECS, see [Deep Learning Containers on Amazon ECS](#) in the *AWS Deep Learning AMI Developer Guide*.

Deep Learning Containers with Elastic Inference on Amazon ECS

AWS Deep Learning Containers provide a set of Docker images for serving models in TensorFlow and Apache MXNet (Incubating) that take advantage of Amazon Elastic Inference accelerators. Amazon ECS provides task definition parameters to attach Elastic Inference accelerators to your containers. When you specify an Elastic Inference accelerator type in your task definition, Amazon ECS manages the lifecycle of, and configuration for, the accelerator. The Amazon ECS service-linked role is required when using this feature. For more information about Elastic Inference accelerators, see [Amazon Elastic Inference Basics](#).

Important

Your Amazon ECS container instances require at least version 1.30.0 of the container agent. For information about checking your agent version and updating to the latest version, see [Updating the Amazon ECS container agent \(p. 448\)](#).

To get started using Deep Learning Containers with Elastic Inference on Amazon ECS, see [Deep Learning Containers with Elastic Inference on Amazon ECS](#) in the *Amazon Elastic Inference Developer Guide*.

Tutorials for Amazon ECS

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

Topics

- [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters \(p. 727\)](#)
- [Tutorial: Creating a cluster with a Fargate Linux task using the AWS CLI \(p. 729\)](#)
- [Tutorial: Creating a cluster with a Fargate Windows task using the AWS CLI \(p. 734\)](#)
- [Tutorial: Creating a cluster with an EC2 task using the AWS CLI \(p. 739\)](#)
- [Tutorial: Using cluster auto scaling with the AWS Management Console \(p. 746\)](#)
- [Tutorial: Specifying sensitive data using Secrets Manager secrets \(p. 764\)](#)
- [Tutorial: Creating a service using Service Discovery \(p. 769\)](#)
- [Tutorial: Creating a service using a blue/green deployment \(p. 778\)](#)
- [Tutorial: Listening for Amazon ECS CloudWatch Events \(p. 787\)](#)
- [Tutorial: Sending Amazon Simple Notification Service alerts for task stopped events \(p. 788\)](#)
- [Tutorial: Using Amazon EFS file systems with Amazon ECS \(p. 790\)](#)
- [Tutorial: Using FSx for Windows File Server file systems with Amazon ECS \(p. 796\)](#)

Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters

Container instances in your clusters need external network access to communicate with the Amazon ECS service endpoint. However, you might have tasks and services that you would like to run in private subnets. Creating a VPC with both public and private subnets provides you the flexibility to launch tasks and services in either a public or private subnet. Tasks and services in the private subnets can access the internet through a NAT gateway. Services in both the public and private subnets can be configured to use a load balancer so that they can still be reached from the public internet.

This tutorial guides you through creating a VPC with two public subnets and two private subnets, which are provided with internet access through a NAT gateway.

Step 1: Create an Elastic IP Address for Your NAT Gateway

A NAT gateway requires an Elastic IP address in your public subnet, but the VPC wizard does not create one for you. Create the Elastic IP address before running the VPC wizard.

To create an Elastic IP address

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the left navigation pane, choose **Elastic IPs**.
3. Choose **Allocate new address, Allocate, Close**.
4. Note the **Allocation ID** for your newly created Elastic IP address; you enter this later in the VPC wizard.

Step 2: Run the VPC Wizard

The VPC wizard automatically creates and configures most of your VPC resources for you.

To run the VPC wizard

1. In the left navigation pane, choose **VPC Dashboard**.
2. Choose **Launch VPC Wizard, VPC with Public and Private Subnets, Select**.
3. For **VPC name**, give your VPC a unique name.
4. For **Elastic IP Allocation ID**, choose the ID of the Elastic IP address that you created earlier.
5. Choose **Create VPC**.
6. When the wizard is finished, choose **OK**. Note the Availability Zone in which your VPC subnets were created. Your additional subnets should be created in a different Availability Zone.

Non-default subnets, such as those created by the VPC wizard, are not auto-assigned public IPv4 addresses. Instances launched in the public subnet must be assigned a public IPv4 address to communicate with the Amazon ECS service endpoint.

To modify your public subnet's IPv4 addressing behavior

1. In the left navigation pane, choose **Subnets**.
2. Select the public subnet for your VPC. By default, the name created by the VPC wizard is **Public subnet**.
3. Choose **Actions, Modify auto-assign IP settings**.
4. Select the **Enable auto-assign public IPv4 address** check box, and then choose **Save**.

Step 3: Create Additional Subnets

The wizard creates a VPC with a single public and a single private subnet in a single Availability Zone. For greater availability, you should create at least one more of each subnet type in a different Availability Zone so that your VPC has both public and private subnets across two Availability Zones.

To create an additional private subnet

1. In the left navigation pane, choose **Subnets**.
2. Choose **Create Subnet**.
3. For **Name tag**, enter a name for your subnet, such as **Private subnet**.
4. For **VPC**, choose the VPC that you created earlier.
5. For **Availability Zone**, choose a different Availability Zone than your original subnets in the VPC.
6. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.3.0/24** for your second private subnet.
7. Choose **Yes, Create**.

To create an additional public subnet

1. In the left navigation pane, choose **Subnets** and then **Create Subnet**.
2. For **Name tag**, enter a name for your subnet, such as **Public subnet**.
3. For **VPC**, choose the VPC that you created earlier.
4. For **Availability Zone**, choose the same Availability Zone as the additional private subnet that you created in the previous procedure.

5. For **IPv4 CIDR block**, enter a valid CIDR block. For example, the wizard creates CIDR blocks in 10.0.0.0/24 and 10.0.1.0/24 by default. You could use **10.0.2.0/24** for your second public subnet.
6. Choose **Yes, Create**.
7. Select the public subnet that you just created and choose **Route Table, Edit**.
8. By default, the main route table is selected. Choose the other available route table so that the **0.0.0.0/0** destination is routed to the internet gateway (**igw-xxxxxxxx**) and choose **Save**.
9. With your second public subnet still selected, choose **Subnet Actions, Modify auto-assign IP settings**.
10. Select **Enable auto-assign public IPv4 address** and choose **Save, Close**.

Next Steps

After you have created your VPC, you should consider the following next steps:

- Create security groups for your public and private resources if they require inbound network access. For more information, see [Working with Security Groups](#) in the *Amazon VPC User Guide*.
- Create Amazon ECS clusters in your private or public subnets. For more information, see [Creating a cluster \(p. 176\)](#). If you use the cluster creation wizard in the Amazon ECS console, you can specify the VPC that you just created and the public or private subnets in which to launch your instances, depending on your use case.
 - To make your containers directly accessible from the internet, launch instances into your *public* subnets. Be sure to configure your container instance security groups appropriately.
 - To avoid making containers directly accessible from the internet, launch instances into your *private* subnets.
- Create a load balancer in your public subnets that can route traffic to services in your public or private subnets. For more information, see [Service load balancing \(p. 574\)](#).

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 \(p. 729\)](#)
- [Step 1: Create a Cluster \(p. 730\)](#)
- [Step 2: Register a Linux Task Definition \(p. 730\)](#)
- [Step 3: List Task Definitions \(p. 731\)](#)
- [Step 4: Create a Service \(p. 732\)](#)
- [Step 5: List Services \(p. 732\)](#)
- [Step 6: Describe the Running Service \(p. 732\)](#)
- [Step 7: Clean Up \(p. 734\)](#)

Prerequisites

This tutorial assumes that the following prerequisites have been completed.

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see [Installing the AWS Command Line Interface](#).
- The steps in [Setting up with Amazon ECS \(p. 7\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Docker Hub so your task must have internet access. To give your task a route to the internet, use one of the following options.
 - Use a private subnet with a NAT gateway that has an elastic IP address.
 - Use a public subnet and assign a public IP address to the task.

For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters](#).

Step 1: Create a Cluster

By default, your account receives a default cluster.

Note

The benefit of using the default cluster that is provided for you is that you don't have to specify the `--cluster` `cluster_name` option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster` `cluster_name` for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{  
  "cluster": {  
    "status": "ACTIVE",  
    "statistics": [],  
    "clusterName": "fargate-cluster",  
    "registeredContainerInstancesCount": 0,  
    "pendingTasksCount": 0,  
    "runningTasksCount": 0,  
    "activeServicesCount": 0,  
    "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"  
  }  
}
```

Step 2: Register a 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 \(p. 194\)](#).

```
{  
  "family": "sample-fargate",  
  "platformFamily": "LINUX",  
  "networkMode": "awsvpc",  
  "containerDefinitions": [  
    {
```

```

    "name": "fargate-app",
    "image": "httpd:2.4",
    "portMappings": [
        {
            "containerPort": 80,
            "hostPort": 80,
            "protocol": "tcp"
        }
    ],
    "essential": true,
    "entryPoint": [
        "sh",
        "-c"
    ],
    "command": [
        "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
<style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' > /usr/local/apache2/htdocs/index.html && httpd-foreground\""
    ]
},
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512"
}

```

The above example JSON can be passed to the AWS CLI in two ways: You can save the task definition JSON as a file and pass it with the `--cli-input-json file://path_to_file.json` option. Or, you can escape the quotation marks in the JSON and pass the JSON container definitions on the command line as in the below example. If you choose to pass the container definitions on the command line, your command additionally requires a `--family` parameter that is used to keep multiple versions of your task definition associated with each other.

To use a JSON file for container definitions:

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/fargate-task.json
```

The `register-task-definition` command returns a description of the task definition after it completes its registration.

Step 3: List Task Definitions

You can list the task definitions for your account at any time with the `list-task-definitions` command. The output of this command shows the `family` and `revision` values that you can use together when calling `run-task` or `start-task`.

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

Output:

```
{
    "taskDefinitionArns": [
        "arn:aws:ecs:region:aws_account_id:task-definition/sample-fargate:1"
    ]
}
```

Step 4: Create a Service

After you have registered a task for your account, you can create a service for the registered task in your cluster. For this example, you create a service with one instance of the `sample-fargate:1` task definition running in your cluster. The task requires a route to the internet, so there are two ways you can achieve this. One way is to use a private subnet configured with a NAT gateway with an elastic IP address in a public subnet. Another way is to use a public subnet and assign a public IP address to your task. We provide both examples below.

Example using a private subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-abcd1234]}"
```

Example using a public subnet.

```
aws ecs create-service --cluster fargate-cluster --service-name fargate-service --task-definition sample-fargate:1 --desired-count 1 --launch-type "FARGATE" --network-configuration "awsvpcConfiguration={subnets=[subnet-abcd1234],securityGroups=[sg-abcd1234],assignPublicIp=ENABLED}"
```

The `create-service` command returns a description of the task definition after it completes its registration.

Step 5: List Services

List the services for your cluster. You should see the service that you created in the previous section. You can take the service name or the full ARN that is returned from this command and use it to describe the service later.

```
aws ecs list-services --cluster fargate-cluster
```

Output:

```
{  
    "serviceArns": [  
        "arn:aws:ecs:region:aws_account_id:service/fargate-service"  
    ]  
}
```

Step 6: Describe the Running Service

Describe the service using the service name retrieved earlier to get more information about the task.

```
aws ecs describe-services --cluster fargate-cluster --services fargate-service
```

If successful, this will return a description of the service failures and services. For example, in services section, you will find information on deployments, such as the status of the tasks as running or pending. You may also find information on the task definition, the network configuration and time-stamped events. In the failures section, you will find information on failures, if any, associated with the call. For troubleshooting, see [Service Event Messages](#). For more information about the service description,

see [Describe Services](#). If your instance was launched in a public subnet, you can view the service task from the internet by using the AWS CLI command `list-tasks` to retrieve the task ID needed for the command `describe-tasks` to retrieve the public IP address of the website.

```
{
    "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-b744699e6cf0) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
        "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
        "createdAt": 1510811364.691
    }
],
"runningCount": 0,
"placementStrategy": []
}
],
"failures": []
}
```

Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

Tutorial: Creating a cluster with 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 \(p. 735\)](#)
- [Step 1: Create a Cluster \(p. 735\)](#)
- [Step 2: Register a Windows Task Definition \(p. 735\)](#)
- [Step 3: List task definitions \(p. 737\)](#)
- [Step 4: Create a service \(p. 737\)](#)
- [Step 5: List services \(p. 737\)](#)

- [Step 6: Describe the Running Service \(p. 738\)](#)
- [Step 7: Clean Up \(p. 739\)](#)

Prerequisites

This tutorial assumes that the following prerequisites have been completed.

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see [Installing the AWS Command Line Interface](#).
- The steps in [Setting up with Amazon ECS \(p. 7\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.
- You have a VPC and security group created to use. This tutorial uses a container image hosted on Docker Hub so your task must have internet access. To give your task a route to the internet, use one of the following options.
 - Use a private subnet with a NAT gateway that has an elastic IP address.
 - Use a public subnet and assign a public IP address to the task.

For more information, see [Tutorial: Creating a VPC with public and private subnets for your clusters](#).

Step 1: Create a Cluster

By default, your account receives a default cluster.

Note

The benefit of using the default cluster that is provided for you is that you don't have to specify the `--cluster` `cluster_name` option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster` `cluster_name` for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name fargate-cluster
```

Output:

```
{  
    "cluster": {  
        "status": "ACTIVE",  
        "statistics": [],  
        "clusterName": "fargate-cluster",  
        "registeredContainerInstancesCount": 0,  
        "pendingTasksCount": 0,  
        "runningTasksCount": 0,  
        "activeServicesCount": 0,  
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/fargate-cluster"  
    }  
}
```

Step 2: Register a 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 \(p. 194\)](#).

```
{
    "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://$HOME/tasks/fargate-task.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-b744699e6cfcc) (task f4c1ad74-a5c6-4620-90cf-2aff118df5fc).",
    "id": "095703e1-0ca3-4379-a7c8-c0f1b8b95ace",
    "createdAt": 1510811364.691
}
],
"runningCount": 0,
"placementStrategy": []
}
],
"failures": []
}
```

Step 7: Clean Up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

Delete the service.

```
aws ecs delete-service --cluster fargate-cluster --service fargate-service --force
```

Delete the cluster.

```
aws ecs delete-cluster --cluster fargate-cluster
```

Tutorial: Creating a cluster with an EC2 task using the AWS CLI

The following steps help you set up a cluster, register a task definition, run a task, and perform other common scenarios in Amazon ECS with the AWS CLI. Ensure that you are using the latest version of

the AWS CLI. For more information on how to upgrade to the latest version, see [Installing the AWS Command Line Interface](#).

Topics

- [Prerequisites \(p. 740\)](#)
- [Step 1: Create a Cluster \(p. 740\)](#)
- [Step 2: Launch an Instance with the Amazon ECS AMI \(p. 741\)](#)
- [Step 3: List Container Instances \(p. 741\)](#)
- [Step 4: Describe your Container Instance \(p. 741\)](#)
- [Step 5: Register a Task Definition \(p. 743\)](#)
- [Step 6: List Task Definitions \(p. 744\)](#)
- [Step 7: Run a Task \(p. 745\)](#)
- [Step 8: List Tasks \(p. 745\)](#)
- [Step 9: Describe the Running Task \(p. 746\)](#)

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading your AWS CLI, see [Installing the AWS Command Line Interface](#).
- The steps in [Setting up with Amazon ECS \(p. 7\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.
- You have a VPC and security group created to use. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters](#).

Step 1: Create a Cluster

By default, your account receives a default cluster when you launch your first container instance.

Note

The benefit of using the default cluster that is provided for you is that you don't have to specify the `--cluster` `cluster_name` option in the subsequent commands. If you do create your own, non-default, cluster, you must specify `--cluster` `cluster_name` for each command that you intend to use with that cluster.

Create your own cluster with a unique name with the following command:

```
aws ecs create-cluster --cluster-name MyCluster
```

Output:

```
{  
    "cluster": {  
        "clusterName": "MyCluster",  
        "status": "ACTIVE",  
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/MyCluster"  
    }  
}
```

```
}
```

Step 2: Launch an Instance with the Amazon ECS AMI

You must have an Amazon ECS container instance in your cluster before you can run tasks on it. If you do not have any container instances in your cluster, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#) for more information.

Step 3: List Container Instances

Within a few minutes of launching your container instance, the Amazon ECS agent registers the instance with your default cluster. You can list the container instances in a cluster by running the following command:

```
aws ecs list-container-instances --cluster default
```

Output:

```
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:aws_account_id:container-instance/container_instance_ID"
    ]
}
```

Step 4: Describe your Container Instance

After you have the ARN or ID of a container instance, you can use the **describe-container-instances** command to get valuable information on the instance, such as remaining and registered CPU and memory resources.

```
aws ecs describe-container-instances --cluster default --container-
instances container_instance_ID
```

Output:

```
{
    "failures": [],
    "containerInstances": [
        {
            "status": "ACTIVE",
            "registeredResources": [
                {
                    "integerValue": 1024,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "CPU",
                    "doubleValue": 0.0
                },
                {
                    "integerValue": 995,
                    "longValue": 0,
                    "type": "INTEGER",
                    "name": "MEMORY",
                    "doubleValue": 0.0
                }
            ]
        }
    ]
}
```

```
{  
    "name": "PORTS",  
    "longValue": 0,  
    "doubleValue": 0.0,  
    "stringSetValue": [  
        "22",  
        "2376",  
        "2375",  
        "51678"  
    ],  
    "type": "STRINGSET",  
    "integerValue": 0  
,  
{  
    "name": "PORTS_UDP",  
    "longValue": 0,  
    "doubleValue": 0.0,  
    "stringSetValue": [],  
    "type": "STRINGSET",  
    "integerValue": 0  
}  
],  
"ec2InstanceId": "instance_id",  
"agentConnected": true,  
"containerInstanceArn": "arn:aws:ecs:us-west-2:aws_account_id:container-  
instance/container_instance_ID",  
"pendingTasksCount": 0,  
"remainingResources": [  
    {  
        "integerValue": 1024,  
        "longValue": 0,  
        "type": "INTEGER",  
        "name": "CPU",  
        "doubleValue": 0.0  
},  
    {  
        "integerValue": 995,  
        "longValue": 0,  
        "type": "INTEGER",  
        "name": "MEMORY",  
        "doubleValue": 0.0  
},  
    {  
        "name": "PORTS",  
        "longValue": 0,  
        "doubleValue": 0.0,  
        "stringSetValue": [  
            "22",  
            "2376",  
            "2375",  
            "51678"  
        ],  
        "type": "STRINGSET",  
        "integerValue": 0  
},  
    {  
        "name": "PORTS_UDP",  
        "longValue": 0,  
        "doubleValue": 0.0,  
        "stringSetValue": [],  
        "type": "STRINGSET",  
        "integerValue": 0  
}  
],  
"runningTasksCount": 0,  
"attributes": [
```

```
{
    "name": "com.amazonaws.ecs.capability.privileged-container"
},
{
    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.17"
},
{
    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
},
{
    "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
},
{
    "name": "com.amazonaws.ecs.capability.logging-driver.json-file"
},
{
    "name": "com.amazonaws.ecs.capability.logging-driver.syslog"
},
],
"versionInfo": {
    "agentVersion": "1.5.0",
    "agentHash": "b197edd",
    "dockerVersion": "DockerVersion: 1.7.1"
}
}
]
```

You can also find the Amazon EC2 instance ID that you can use to monitor the instance in the Amazon EC2 console or with the `aws ec2 describe-instances --instance-id instance_id` command.

Step 5: Register a Task Definition

Before you can run a task on your ECS cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses a `busybox` image from Docker Hub and simply sleeps for 360 seconds. For more information about the available task definition parameters, see [Amazon ECS task definitions \(p. 194\)](#).

```
{
    "containerDefinitions": [
        {
            "name": "sleep",
            "image": "busybox",
            "cpu": 10,
            "command": [
                "sleep",
                "360"
            ],
            "memory": 10,
            "essential": true
        }
    ],
    "family": "sleep360"
}
```

The above example JSON can be passed to the AWS CLI in two ways: You can save the task definition JSON as a file and pass it with the `--cli-input-json file://path_to_file.json` option. Or, you can escape the quotation marks in the JSON and pass the JSON container definitions on the command line as in the below example. If you choose to pass the container definitions on the command line, your command additionally requires a `--family` parameter that is used to keep multiple versions of your task definition associated with each other.

To use a JSON file for container definitions:

```
aws ecs register-task-definition --cli-input-json file://$HOME/tasks/sleep360.json
```

To use a JSON string for container definitions:

```
aws ecs register-task-definition --family sleep360 --container-definitions "[{"name": "sleep", "image": "busybox", "cpu": 10, "command": ["sleep", "360"], "memory": 10, "essential": true}]"
```

The **register-task-definition** returns a description of the task definition after it completes its registration.

```
{
    "taskDefinition": {
        "volumes": [],
        "taskDefinitionArn": "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/sleep360:1",
        "containerDefinitions": [
            {
                "environment": [],
                "name": "sleep",
                "mountPoints": [],
                "image": "busybox",
                "cpu": 10,
                "portMappings": [],
                "command": [
                    "sleep",
                    "360"
                ],
                "memory": 10,
                "essential": true,
                "volumesFrom": []
            }
        ],
        "family": "sleep360",
        "revision": 1
    }
}
```

Step 6: List Task Definitions

You can list the task definitions for your account at any time with the **list-task-definitions** command. The output of this command shows the **family** and **revision** values that you can use together when calling **run-task** or **start-task**.

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

Output:

```
{
    "taskDefinitionArns": [
        "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/sleep300:1",
        "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/sleep300:2",
        "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/sleep360:1",
        "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/wordpress:3",
        "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/wordpress:4",
        "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/wordpress:5",
        "arn:aws:ec2:us-east-1:$aws_account_id:task-definition/wordpress:6"
    ]
}
```

```
        "arn:aws:ec2:us-east-1:aws_account_id:task-definition/wordpress:6"
    ]  
}
```

Step 7: Run a Task

After you have registered a task for your account and have launched a container instance that is registered to your cluster, you can run the registered task in your cluster. For this example, you place a single instance of the `sleep360:1` task definition in your default cluster.

```
aws ecs run-task --cluster default --task-definition sleep360:1 --count 1
```

Output:

```
{
  "tasks": [
    {
      "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
      "overrides": {
        "containerOverrides": [
          {
            "name": "sleep"
          }
        ],
        "lastStatus": "PENDING",
        "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-
instance/container_instance_ID",
        "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/default",
        "desiredStatus": "RUNNING",
        "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/
sleep360:1",
        "containers": [
          {
            "containerArn": "arn:aws:ecs:us-
east-1:aws_account_id:container/container_ID",
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",
            "lastStatus": "PENDING",
            "name": "sleep"
          }
        ]
      }
    }
  ]
}
```

Step 8: List Tasks

List the tasks for your cluster. You should see the task that you ran in the previous section. You can take the task ID or the full ARN that is returned from this command and use it to describe the task later.

```
aws ecs list-tasks --cluster default
```

Output:

```
{
  "taskArns": [
    "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID"
```

```
    ]  
}
```

Step 9: Describe the Running Task

Describe the task using the task ID retrieved earlier to get more information about the task.

```
aws ecs describe-tasks --cluster default --task task_ID
```

Output:

```
{  
    "failures": [],  
    "tasks": [  
        {  
            "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",  
            "overrides": {  
                "containerOverrides": [  
                    {  
                        "name": "sleep"  
                    }  
                ]  
            },  
            "lastStatus": "RUNNING",  
            "containerInstanceArn": "arn:aws:ecs:us-east-1:aws_account_id:container-  
instance/container_instance_ID",  
            "clusterArn": "arn:aws:ecs:us-east-1:aws_account_id:cluster/default",  
            "desiredStatus": "RUNNING",  
            "taskDefinitionArn": "arn:aws:ecs:us-east-1:aws_account_id:task-definition/  
sleep360:1",  
            "containers": [  
                {  
                    "containerArn": "arn:aws:ecs:us-  
east-1:aws_account_id:container/container_ID",  
                    "taskArn": "arn:aws:ecs:us-east-1:aws_account_id:task/task_ID",  
                    "lastStatus": "RUNNING",  
                    "name": "sleep",  
                    "networkBindings": []  
                }  
            ]  
        }  
    ]  
}
```

Tutorial: Using cluster auto scaling with the AWS Management Console

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

Topics

- [Prerequisites \(p. 747\)](#)
- [Step 1: Create an Amazon ECS cluster \(p. 747\)](#)
- [Step 2: Create the Auto Scaling resources \(p. 747\)](#)

- Step 3: Create a capacity provider (p. 749)
- Step 4: Set a default capacity provider strategy for the cluster (p. 761)
- Step 5: Register a task definition (p. 761)
- Step 6: Run a task (p. 762)
- Step 7: Verify (p. 762)
- Step 8: Clean up (p. 763)

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in [Setting up with Amazon ECS \(p. 7\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.
- The Amazon ECS container instance IAM role is created. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
- The Amazon ECS service-linked IAM role is created. For more information, see [Service-linked role for Amazon ECS \(p. 685\)](#).
- The Auto Scaling service-linked IAM role is created. For more information, see [Service-Linked Roles for Amazon EC2 Auto Scaling](#) in the *Amazon EC2 Auto Scaling User Guide*.
- You have a VPC and security group created to use. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters](#).

Step 1: Create an Amazon ECS cluster

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

To create an empty cluster

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

Step 2: Create the Auto Scaling resources

Use the following steps to create an Amazon EC2 launch template and Auto Scaling group.

To create an Amazon EC2 launch template

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.

2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. On the navigation pane, under **Instances**, choose **Launch Templates**.
4. On the next page, choose **Create launch template**.
5. On the **Create launch template** page, complete the following steps.
 - a. For **Launch template name**, enter `ConsoleTutorial-LaunchTemplate`.
 - b. For **Template version description**, provide a description for the launch template.
 - c. For **Amazon Machine Image (AMI)**, search for the latest Amazon ECS-optimized AMI. The AMI ID can be retrieved using the following link: [View AMI ID](#). For more information, see [Retrieving Amazon ECS-Optimized AMI metadata \(p. 339\)](#).
 - d. For **Instance type**, select an instance type. For the purposes of this tutorial, the `t2.micro` instance type works.
 - e. For **Security groups**, select one or more security groups.
 - f. Expand the **Advanced Details** section to specify the IAM instance profile and user data for your Amazon ECS container instances.
 - i. For **IAM instance profile**, select your container instance IAM role. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).
 - ii. For **User data**, paste the following script into the field. The `ConsoleTutorial-cluster` cluster was created in the first step.

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

6. Choose **Create launch template**.

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

To create an Auto Scaling group

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. On the navigation pane, under **Auto Scaling**, choose **Auto Scaling Groups, Create Auto Scaling group**.
4. On the **Create Auto Scaling group** page, complete the following steps.
 - a. For **Auto Scaling group name**, enter `ConsoleTutorial-ASG` for the Auto Scaling group name.
 - b. For **Launch template**, select the `ConsoleTutorial-LaunchTemplate` launch template.
 - c. Review the contents of the launch template, then choose **Next**.
 - d. On the **Configure settings** step, under **Network**, select a **VPC** and one or more **Subnets** to use and then choose **Next**.
 - e. On the **Configure advanced options** step, choose **Next**.
 - f. On the **Configure group size and scaling policies** step, for **Desired capacity**, enter 0. This tutorial uses Amazon ECS managed scaling so there is no need to have the Auto Scaling group launch any initial instances. For **Minimum capacity**, enter 0. For **Maximum capacity**, enter 2.
 - g. For **Instance scale-in protection**, select **Enable instance scale-in protection** and then choose **Next**. This enables you to use managed termination protection for the instances in the Auto Scaling group, which prevents your container instances that contain tasks from being terminated during scale-in actions.
 - h. On the **Add notifications and Add tags** steps, choose **Next**.
 - i. On the **Review** step, review the Auto Scaling group settings and then choose **Create Auto Scaling group**.

5. Repeat steps 3 and 4 to create a second Auto Scaling group. For **Auto Scaling group name** use `ConsoleTutorial-ASG-burst` and when setting the group size, set the **Desired capacity** and **Minimum capacity** to 0 and the Maximum capacity to 20.

Step 3: Create a capacity provider

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

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. 180).

For Amazon ECS on Amazon EC2 users, a capacity provider consists of a capacity provider name, an Auto Scaling group, and the settings for managed scaling and managed termination protection. With managed scaling, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group which provides auto scaling for your cluster's infrastructure. For more information, see Auto Scaling group capacity providers (p. 184).

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 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 managed termination protection is enabled, managed scaling must also be enabled 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 enable 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. 178).

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 support for Windows containers on Fargate.
- 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. 182).
- 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 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) or 1.0.0 or later (for Windows). For more information, see AWS Fargate platform versions (p. 169).
- 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. 636).
- 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. 237).

- 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"  
  
        ]  
  
    }  
}
```

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=true}"
\

--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. 181).

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=true}"
\

--desired-count integer \
--region us-west-2
```

Auto Scaling group capacity providers

Amazon ECS capacity providers can use Auto Scaling groups to manage the Amazon EC2 instances registered to their clusters. You can use the managed scaling feature to have Amazon ECS manage the scale-in and scale-out actions of the Auto Scaling group or you can manage the scaling actions yourself. For more information, see [Amazon ECS cluster auto scaling](#) (p. 189).

Topics

- [Auto Scaling group capacity providers considerations](#) (p. 184)
- [Creating an Auto Scaling group](#) (p. 184)
- [Creating an Auto Scaling group capacity provider](#) (p. 185)
- [Updating an Auto Scaling group capacity provider](#) (p. 186)
- [Creating a cluster with an Auto Scaling group capacity provider](#) (p. 187)
- [Deleting an Auto Scaling group capacity provider](#) (p. 188)

Auto Scaling group capacity providers considerations

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

- It is recommended that you create a new empty Auto Scaling group to use with a capacity provider rather than using an existing one. If you use an existing Auto Scaling group,

any Amazon EC2 instances associated with the group that were already running and

registered to an Amazon ECS cluster prior to the Auto Scaling group being used to create a capacity provider may not be properly registered with the capacity provider. This may cause issues when using the capacity provider in a capacity provider strategy. The `DescribeContainerInstances` API can confirm whether a container instance is associated with a capacity provider or not.

Note

To create an empty Auto Scaling group, set the desired count to zero. After you have created the capacity provider and associated it with a cluster, you can then scale it out.

- An Auto Scaling group must have a `MaxSize` greater than zero to enable it to scale out.
- If the Auto Scaling group is unable to scale out to accommodate the number of tasks run, the tasks will fail to transition beyond the `PROVISIONING` state.
- When using managed termination protection, managed scaling must be enabled otherwise managed termination protection will not work.
- When using managed scaling, the Auto Scaling group shouldn't have any scaling policies attached to it other than the ones Amazon ECS creates, otherwise the Amazon ECS created scaling plans will receive an `ActiveWithProblems` error. For more information, see [Avoiding the ActiveWithProblems error in the AWS Auto Scaling User Guide](#).

Creating an Auto Scaling group

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

Note

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

For more information on replacing an Auto Scaling launch configuration with an Amazon EC2 launch template, see [Replacing a launch configuration with a launch template in the Amazon EC2 Auto Scaling User Guide](#)

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

- If managed termination protection is enabled when you create a capacity provider, the Auto Scaling group and each Amazon EC2 instance in the Auto Scaling group must have

instance protection from scale in enabled as well. For more information, see [Instance Protection in the AWS Auto Scaling User Guide](#).

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

For more information on creating an Amazon EC2 Auto Scaling launch template, see [Launch Templates in the Amazon EC2 Auto Scaling User Guide](#).

For more information on creating an Amazon EC2 Auto Scaling group, see [Auto Scaling groups in the Amazon EC2 Auto Scaling User Guide](#).

Creating an Auto Scaling group capacity provider

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

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

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

To create an Auto Scaling group capacity provider (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region your cluster exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose **Capacity Providers**, and then choose **Create**.
6. For **Capacity provider name**, enter a capacity provider name.
7. For **Auto Scaling group**, select the Auto Scaling group to associate with the capacity provider. The Auto Scaling group must already be created. For more information, see Creating an Auto Scaling group (p. 184).
8. For **Managed scaling**, choose your managed scaling option. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling plans. When managed scaling is disabled, you manage your Auto Scaling groups yourself.
9. For **Target capacity %**, if managed scaling is enabled, specify an integer between 1 and 100. The target capacity value is used as the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy. This target capacity value is matched on a best effort basis. For example, a value of 100 will result in the Amazon EC2 instances in your Auto Scaling group being completely utilized and any instances not running any tasks will be scaled in, but this behavior is not guaranteed at all times.
10. For **Managed termination protection**, choose your managed termination protection option. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled and if managed scaling is enabled. Managed termination protection is only supported on standalone tasks or tasks in a service using the replica scheduling strategy. For tasks in a service using the daemon scheduling strategy, the instances are not protected.
11. Choose **Create** to complete the capacity provider creation.

To create an Auto Scaling group capacity provider (AWS CLI)

- Use the following command to create a new capacity provider.

- [create-capacity-provider \(AWS CLI\)](#)

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

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

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

Updating an Auto Scaling group capacity provider

A capacity provider can be updated to change its managed scaling and managed termination protection settings. Use the following steps to update an existing capacity provider.

To update an Auto Scaling group capacity provider (AWS Management Console)

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region the cluster the capacity provider is associated with exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : name** page, choose the **Capacity Providers** tab.
6. Select the capacity provider to update and choose **Update**.
7. On the **Update Capacity Provider** page, the following parameters can be updated.
 - a. For **Managed scaling**, choose your managed scaling option. When managed scaling is enabled, Amazon ECS manages the scale-in and scale-out actions of the Auto Scaling group through the use of AWS Auto Scaling scaling plans. When managed scaling is disabled, you manage your Auto Scaling groups yourself.
 - b. For **Target capacity %**, if managed scaling is enabled, specify an integer between 1 and 100. The target capacity value is used as the target value for the CloudWatch metric used in the Amazon ECS-managed target tracking scaling policy. This target capacity value is matched on a best effort basis. For example, a value of 100 will result in the Amazon EC2 instances in your Auto Scaling group being completely utilized and any instances not running any tasks will be scaled in, but this behavior is not guaranteed at all times.
 - c. For **Managed termination protection**, choose your managed termination protection option. When managed termination protection is enabled, Amazon ECS prevents Amazon EC2 instances that contain tasks and that are in an Auto Scaling group from being terminated during a scale-in action. Managed termination protection can only be enabled if the Auto Scaling group also has instance protection from scale in enabled and if managed scaling is enabled. Managed termination protection is only supported on standalone tasks or tasks in a service using the replica scheduling strategy. For tasks in a service using the daemon scheduling strategy, the instances are not protected.
8. Choose **Update** to request capacity provider update.

9. To verify whether the capacity provider update was successful, check the **Update Status** column on the **Capacity Providers** tab.

To update an Auto Scaling group capacity provider (AWS CLI)

- Use the following command to create a new capacity provider.
- **update-capacity-provider (AWS CLI)**

```
aws ecs update-capacity-provider \
    --name CapacityProviderName \
    --auto-scaling-group-provider managedScaling=\{status='ENABLED' / \
    DISABLED',targetCapacity=integer,minimumScalingStepSize=integer,maximumScalingStepSize=integer \
    \ \
    --region us-east-2
```

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

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

Creating a cluster with an Auto Scaling group capacity provider

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

For AWS Management Console steps, see [Creating a cluster \(p. 176\)](#).

To create a cluster with an Auto Scaling group capacity provider (AWS CLI)

Use the following command to create a new cluster and associate one or more capacity providers with it.

- **create-cluster (AWS CLI)**

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

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

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

Deleting an Auto Scaling group capacity provider

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

Prior to an Auto Scaling group capacity provider being deleted, the capacity provider must be removed from the capacity provider strategy from all services. The `UpdateService` API or the update service workflow in the AWS Management Console can be used to remove a capacity provider from a service's capacity provider strategy. The force new deployment option can be used to ensure that any tasks using the Amazon EC2 instance capacity provided by the capacity provider are transitioned to use the capacity from the remaining capacity providers.

There are other prerequisites that must be performed to delete a capacity provider but they are specific to the tool used and are mentioned in the following steps.

Use the following steps to delete an Auto Scaling group capacity provider.

To delete an Auto Scaling group capacity provider (AWS Management Console)

When deleting a capacity provider using the AWS Management Console, the console goes through two steps. The capacity provider is first disassociated from the cluster completely and then it is deleted. In rare cases, the capacity provider may be successfully disassociated from the cluster but is unable to be deleted. In those cases, you must use either the Amazon ECS API or the AWS CLI to view the status of the capacity provider and delete it.

Note

Only capacity providers that are currently associated with a cluster are visible in the AWS Management Console. To delete a capacity provider that is not associated with a cluster, you must use the Amazon ECS API, SDK, or AWS CLI.

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region your cluster exists in.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, select your cluster.
5. On the **Cluster : *name*** page, choose the **Capacity Providers** tab.
6. Select the capacity provider you want to delete and then choose **Delete**.

To delete an Auto Scaling group capacity provider (AWS CLI)

When using the AWS CLI to delete a capacity provider, the capacity provider must first be disassociated from the cluster. The following options are available to disassociate a capacity provider from a cluster.

Option 1: Use the `delete` command to delete the cluster. This will disassociate the capacity provider from the cluster upon successful deletion of the cluster.

- `delete-cluster` (AWS CLI)

```
aws ecs delete-cluster \
--cluster MyCluster
```

Option 2: Use the **put-cluster-capacity-providers** command to disassociate a capacity provider from a cluster. If you have other capacity providers associated with the cluster that you want to have remain associated with the cluster, you must include those when using the command.

The following example will remove all existing capacity providers from the specified cluster.

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

```
aws ecs put-cluster-capacity-providers \
    --cluster MyCluster \
    --capacity-providers [] \
    --default-capacity-provider-strategy []
```

Use the **delete-capacity-provider** command to delete a capacity provider. You can specify the capacity provider using its short name or the full Amazon Resource Name (ARN).

- **delete-capacity-provider** (AWS CLI)

Example using the short name:

```
aws ecs delete-capacity-provider \
    --capacity-provider ExampleCapacityProvider
```

Example using the full ARN:

```
aws ecs delete-capacity-provider \
    --capacity-provider arn:aws:ecs:us-west-2:123456789012:capacity-
    provider/ExampleCapacityProvider
```

(p.) for more information.

To create a capacity provider

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

Step 4: Set a default capacity provider strategy for the cluster

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

To define a default capacity provider strategy

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

Step 5: Register a task definition

Before you can run a task on your cluster, you must register a task definition. Task definitions are lists of containers grouped together. The following example is a simple task definition that uses an `amazonlinux` image from Docker Hub and simply sleeps. For more information about the available task definition parameters, see [Amazon ECS task definitions \(p. 194\)](#).

To register a task definition

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

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

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

Step 6: Run a task

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

To run a task

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Task Definitions**.
4. Select your **ConsoleTutorial-taskdef** task definition.
5. From the **Actions** menu, choose **Run Task**.
6. Use the following steps to complete the run task workflow.
 - a. For **Capacity provider strategy**, the default capacity provider strategy for the cluster must be selected.
 - b. For **Cluster**, select your **ConsoleTutorial-cluster** cluster.
 - c. For **Number of tasks**, enter 5.
 - d. For **Placement Templates**, choose **BinPack**.
 - e. Choose **Run Task**.

Step 7: Verify

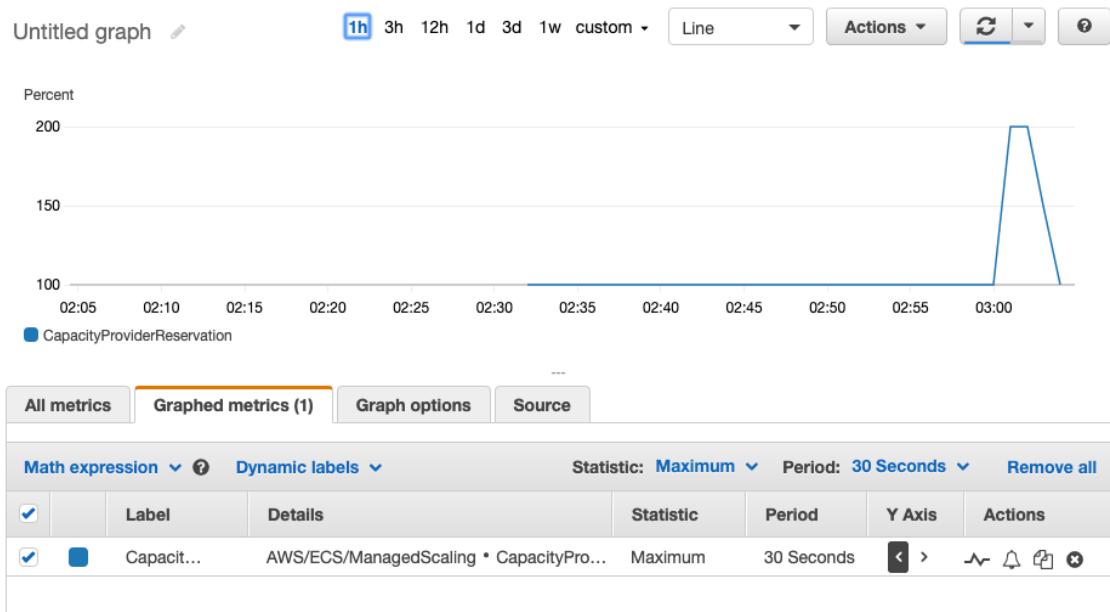
At this point in the tutorial, you should have two Auto Scaling groups with one capacity provider for each of them. The capacity providers have Amazon ECS managed scaling enabled. A cluster was created and five tasks are running.

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

To view the CloudWatch metrics for your cluster

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

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



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

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

To verify the Auto Scaling group scaled out

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

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

To verify the Auto Scaling group scaled out

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

Step 8: Clean up

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

To clean up the tutorial resources

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

Tutorial: 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 \(p. 303\)](#).

The following tutorial shows how to create an Secrets Manager secret, reference the secret in an Amazon ECS task definition, and then verify it worked by querying the environment variable inside a container showing the contents of the secret.

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The steps in [Setting up with Amazon ECS \(p. 7\)](#) have been completed.
- Your AWS user has the required IAM permissions to create the Secrets Manager and Amazon ECS resources described.

Step 1: Create an Secrets Manager secret

You can use the Secrets Manager console to create a secret for your sensitive data. In this tutorial we will be creating a basic secret for storing a username and password to reference later in a container. For more information, see [Creating a Basic Secret](#) in the *AWS Secrets Manager User Guide*.

To create a basic secret

Use Secrets Manager to create a secret for your sensitive data.

1. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose **Store a new secret**.
3. For **Select secret type**, choose **Other type of secrets**.
4. For **Specify the key/value pairs to be stored in this secret**, choose the **Plaintext** tab and replace the existing text with the following text. The text value you specify here will be the environment variable value in your container at the end of the tutorial.

```
password_value
```

5. Choose **Next**.
6. For **Secret name**, type `username_value` and choose **Next**. The secret name value you specify here will be the environment variable name in your container at the end of the tutorial.
7. For **Configure automatic rotation**, leave **Disable automatic rotation** selected and choose **Next**.
8. Review these settings, and then choose **Store** to save everything you entered as a new secret in Secrets Manager.
9. Select the secret you just created and save the **Secret ARN** to reference in your task execution IAM policy and task definition in later steps.

Step 2: Update your task execution IAM role

In order for Amazon ECS to retrieve the sensitive data from your Secrets Manager secret, you must have the Amazon ECS task execution role and reference it in your task definition. This allows the container agent to pull the necessary Secrets Manager resources. If you have not already created your task execution IAM role, see [Amazon ECS task execution IAM role \(p. 691\)](#).

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:  
            ]  
        }  
    ]  
}
```

```

        "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**.

Important

The value for the secret name in the task definition must match the name you specified for the secret name when the secret was created.

```

{
    "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecstaskExecutionRole",
    "containerDefinitions": [
        {
            "entryPoint": [
                "sh",
                "-c"
            ],
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ],
            "command": [
                "/bin/sh -c \\"echo '<html> <head> <title>Amazon ECS Sample
App</title> <style>body {margin-top: 40px; background-color: #333;} </style> </
head><body> <div style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1>
<h2>Congratulations!</h2> <p>Your application is now running on a container in Amazon
ECS.</p> </div></body></html>' > /usr/local/apache2/htdocs/index.html && httpd-
foreground\""
            ],
            "cpu": 10,
            "secrets": [
                {
                    "valueFrom":
"arn:aws:secretsmanager:region:aws_account_id:secret:username_value-u9bH6K",
                    "name": "username_value"
                }
            ],
            "memory": 300,
        }
    ]
}

```

```
        "image": "httpd:2.4",
        "essential": true,
        "name": "ecs-secrets-container"
    }
],
"family": "ecs-secrets-tutorial"
}
```

5. Review the settings and then choose **Create**.

Step 4: Create an Amazon ECS cluster

You can use the Amazon ECS console to create a cluster containing a container instance to run the task on. If you have an existing cluster with at least one container instance registered to it with the available resources to run one instance of the task definition created for this tutorial you can skip to the next step.

For this tutorial we will be creating a cluster with one `t2.micro` container instance using the Amazon ECS-optimized Amazon Linux 2 AMI.

To create a cluster

Use the Amazon ECS console to create a cluster and register one container instance to it.

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. From the navigation bar, select the Region that contains both the Secrets Manager secret and the Amazon ECS task definition you created.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. For **EC2 instance type**, choose `t2.micro`.
6. For **Key pair**, choose a key pair to add to the container instance.

Important

A key pair is required to complete the tutorial, so if you do not already have a key pair created follow the link to the EC2 console to create one.

7. Leave all other fields at their default values and choose **Create**.

Step 5: Run an Amazon ECS task

You can use the Amazon ECS console to run a task using the task definition you created. For this tutorial we will be running a task using the EC2 launch type, using the cluster we created in the previous step.

To run a task

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Task Definitions** and select the `ecs-secrets-tutorial` task definition we created.
3. Select the latest revision of the task definition and then choose **Actions, Run Task**.
4. For **Launch Type**, choose **EC2**.
5. For **Cluster**, choose the `ecs-secrets-tutorial` cluster we created in the previous step.
6. For **Task tagging configuration**, deselect **Enable ECS managed tags**. They are unnecessary for the purposes of this tutorial.
7. Review your task information and choose **Run Task**.

Note

If your task moves from PENDING to STOPPED, or if it displays a PENDING status and then disappears from the listed tasks, your task may be stopping due to an error. For more information, see [Checking stopped tasks for errors \(p. 815\)](#) 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 `ECSecretsTutorial`. Choose **Remove** to confirm the removal of the inline policy.
9. Open the Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
10. Select the `username_value` secret you created and choose **Actions**, **Delete secret**.

Tutorial: Creating a service using Service Discovery

Service discovery has been integrated into the Create Service wizard in the Amazon ECS console. For more information, see [Creating an Amazon ECS service \(p. 546\)](#).

The following tutorial shows how to create an ECS service containing a Fargate task that uses service discovery with the AWS CLI.

For a list of Regions that support service discovery, see [Service Discovery \(p. 599\)](#).

For information about the Regions that support Fargate, see [the section called “AWS Fargate Regions” \(p. 614\)](#).

Prerequisites

This tutorial assumes that the following prerequisites have been completed:

- The latest version of the AWS CLI is installed and configured. For more information, see [Installing the AWS Command Line Interface](#).
- The steps in [Setting up with Amazon ECS \(p. 7\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.
- You have a VPC and security group created to use. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters](#).

Step 1: Create the Service Discovery resources

Use the following steps to create your service discovery namespace and service discovery service.

To create the Service Discovery resources

1. Create a private service discovery namespace named `tutorial` within one of your existing VPCs:

```
aws servicediscovery create-private-dns-namespace --name tutorial --vpc vpc-abcd1234 --region us-east-1
```

Output:

```
{  
    "OperationId": "h2qe3s6dxftvvvt7riu6lfy2f6c3jlfh4-je6chs2e"  
}
```

2. Using the `OperationId` from the previous output, verify that the private namespace was created successfully. Copy the namespace ID as it is used in subsequent commands.

```
aws servicediscovery get-operation --operation-id h2qe3s6dxftvvvt7riu6lfy2f6c3jlfh4-je6chs2e
```

Output:

```
{  
    "Operation": {  
        "Id": "h2qe3s6dxftvvvt7riu6lfy2f6c3jlfh4-je6chs2e",  
        "Type": "CREATE_NAMESPACE",  
        "Status": "SUCCESS",  
        "CreateDate": 1519777852.502,  
        "UpdateDate": 1519777856.086,  
        "Targets": {  
            "NAMESPACE": "ns-uejictsjen2i4eeg"  
        }  
    }  
}
```

3. Using the `NAMESPACE` ID from the previous output, create a service discovery service named `myapplication`. Copy the service discovery service ID as it is used in subsequent commands:

```
aws servicediscovery create-service --name myapplication --dns-config 'NamespaceId="ns-uejictsjen2i4eeg",DnsRecords=[{Type="A",TTL="300"}]' --health-check-custom-config FailureThreshold=1 --region us-east-1
```

Output:

```
{  
    "Service": {  
        "Id": "srv-utcrh6wavdkggqtk",  
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkggqtk",  
        "Name": "myapplication",  
        "DnsConfig": {  
            "NamespaceId": "ns-uejictsjen2i4eeg",  
            "DnsRecords": [  
                {  
                    "Type": "A",  
                    "TTL": 300  
                }  
            ]  
        },  
        "HealthCheckCustomConfig": {  
            "FailureThreshold": 1  
        },  
        "CreatorRequestId": "e49a8797-b735-481b-a657-b74d1d6734eb"  
    }  
}
```

}

Step 2: Create the Amazon ECS resources

Use the following steps to create your Amazon ECS cluster, task definition, and service.

To create Amazon ECS resources

1. Create an Amazon ECS cluster named `tutorial` to use:

```
aws ecs create-cluster --cluster-name tutorial --region us-east-1
```

Output:

```
{  
    "cluster": {  
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/tutorial",  
        "clusterName": "tutorial",  
        "status": "ACTIVE",  
        "registeredContainerInstancesCount": 0,  
        "runningTasksCount": 0,  
        "pendingTasksCount": 0,  
        "activeServicesCount": 0,  
        "statistics": []  
    }  
}
```

2. Register a task definition that is compatible with Fargate. It requires the use of the `awsvpc` network mode. The following is the example task definition used for this tutorial.

First, create a file named `fargate-task.json` with the contents of the following task definition:

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

```

        ],
        "requiresCompatibilities": [
            "FARGATE"
        ],
        "cpu": "256",
        "memory": "512"
    }
}

```

Then, register the task definition using the `fargate-task.json` file that you created:

```

aws ecs register-task-definition --cli-input-json file://fargate-task.json --region us-east-1

```

3. Create a file named `ecs-service-discovery.json` with the contents of the ECS service that you are going to create. This example uses the task definition created in the previous step. An `awsvpcConfiguration` is required because the example task definition uses the `awsvpc` network mode.

```

{
    "cluster": "tutorial",
    "serviceName": "ecs-service-discovery",
    "taskDefinition": "tutorial-task-def",
    "serviceRegistries": [
        {
            "registryArn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkggqtk"
        }
    ],
    "launchType": "FARGATE",
    "platformVersion": "LATEST",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "assignPublicIp": "ENABLED",
            "securityGroups": [ "sg-abcd1234" ],
            "subnets": [ "subnet-abcd1234" ]
        }
    },
    "desiredCount": 1
}

```

Create your ECS service, specifying the Fargate launch type and the `LATEST` platform version, which supports service discovery:

```

aws ecs create-service --cli-input-json file://ecs-service-discovery.json --region us-east-1

```

Output:

```

{
    "service": {
        "serviceArn": "arn:aws:ecs:region:aws_account_id:service/ecs-service-discovery",
        "serviceName": "ecs-service-discovery",
        "clusterArn": "arn:aws:ecs:region:aws_account_id:cluster/tutorial",
        "loadBalancers": [],
        "serviceRegistries": [
            {
                "registryArn": "arn:aws:servicediscovery:region:aws_account_id:service/srv-utcrh6wavdkggqtk"
            }
        ]
    }
}

```

```
        ],
        "status": "ACTIVE",
        "desiredCount": 1,
        "runningCount": 0,
        "pendingCount": 0,
        "launchType": "FARGATE",
        "platformVersion": "LATEST",
        "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/tutorial-
task-def:1",
        "deploymentConfiguration": {
            "maximumPercent": 200,
            "minimumHealthyPercent": 100
        },
        "deployments": [
            {
                "id": "ecs-svc/9223370516993140842",
                "status": "PRIMARY",
                "taskDefinition": "arn:aws:ecs:region:aws_account_id:task-definition/
tutorial-task-def:1",
                "desiredCount": 1,
                "pendingCount": 0,
                "runningCount": 0,
                "createdAt": 1519861634.965,
                "updatedAt": 1519861634.965,
                "launchType": "FARGATE",
                "platformVersion": "1.1.0",
                "networkConfiguration": {
                    "awsvpcConfiguration": {
                        "subnets": [
                            "subnet-abcd1234"
                        ],
                        "securityGroups": [
                            "sg-abcd1234"
                        ],
                        "assignPublicIp": "ENABLED"
                    }
                }
            }
        ],
        "roleArn": "arn:aws:iam::aws_account_id:role/ECSServiceLinkedRole",
        "events": [],
        "createdAt": 1519861634.965,
        "placementConstraints": [],
        "placementStrategy": [],
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "subnets": [
                    "subnet-abcd1234"
                ],
                "securityGroups": [
                    "sg-abcd1234"
                ],
                "assignPublicIp": "ENABLED"
            }
        }
    }
}
```

Step 3: Verify Service Discovery

You can verify that everything has been created properly by querying your service discovery information. After service discovery is configured, you can query it using either the AWS Cloud Map API operations or by using `dig` from within your VPC, as described below.

To verify service discovery configuration

1. Using the service discovery service ID, list the service discovery instances:

```
aws servicediscovery list-instances --service-id srv-utcrh6wavdkggqtk --region us-east-1
```

Output:

```
{  
    "Instances": [  
        {  
            "Id": "16becc26-8558-4af1-9fbd-f81be062a266",  
            "Attributes": {  
                "AWS_INSTANCE_IPV4": "172.31.87.2"  
                "AWS_INSTANCE_PORT": "80",  
                "AVAILABILITY_ZONE": "us-east-1a",  
                "REGION": "us-east-1",  
                "ECS_SERVICE_NAME": "ecs-service-discovery",  
                "ECS_CLUSTER_NAME": "tutorial",  
                "ECS_TASK_DEFINITION_FAMILY": "tutorial-task-def"  
            }  
        }  
    ]  
}
```

2. Using the service discovery namespace and service, use additional parameters to query the details about the service discovery instances:

```
aws servicediscovery discover-instances --namespace-name tutorial --service-name myapplication --query-parameters ECS_CLUSTER_NAME=tutorial --region us-east-1
```

Output:

```
{  
    "Instances": [  
        {  
            "InstanceId": "16becc26-8558-4af1-9fbd-f81be062a266",  
            "NamespaceName": "tutorial",  
            "ServiceName": "ecs-service-discovery",  
            "HealthStatus": "HEALTHY",  
            "Attributes": {  
                "AWS_INSTANCE_IPV4": "172.31.87.2"  
                "AWS_INSTANCE_PORT": "80",  
                "AVAILABILITY_ZONE": "us-east-1a",  
                "REGION": "us-east-1",  
                "ECS_SERVICE_NAME": "ecs-service-discovery",  
                "ECS_CLUSTER_NAME": "tutorial",  
                "ECS_TASK_DEFINITION_FAMILY": "tutorial-task-def"  
            }  
        }  
    ]  
}
```

3. The DNS records created in the Route 53 hosted zone for the service discovery service can be queried with the following AWS CLI commands.

Using the namespace ID, get information about the namespace, which includes the Route 53 hosted zone ID:

```
aws servicediscovery get-namespace --id ns-uejictsjen2i4eeg --region us-east-1
```

Output:

```
{  
    "Namespace": {  
        "Id": "ns-uejictsjen2i4eeg",  
        "Arn": "arn:aws:servicediscovery:region:aws_account_id:namespace/ns-  
uejictsjen2i4eeg",  
        "Name": "tutorial",  
        "Type": "DNS_PRIVATE",  
        "Properties": {  
            "DnsProperties": {  
                "HostedZoneId": "Z35JQ4ZFDRYPLV"  
            }  
        },  
        "CreateDate": 1519777852.502,  
        "CreatorRequestId": "9049a1d5-25e4-4115-8625-96dbda9a6093"  
    }  
}
```

4. Using the Route 53 hosted zone ID, get the resource record set for the hosted zone:

```
aws route53 list-resource-record-sets --hosted-zone-id Z35JQ4ZFDRYPLV --region us-  
east-1
```

Output:

```
{  
    "ResourceRecordSets": [  
        {  
            "Name": "tutorial.",  
            "Type": "NS",  
            "TTL": 172800,  
            "ResourceRecords": [  
                {  
                    "Value": "ns-1536.awsdns-00.co.uk."  
                },  
                {  
                    "Value": "ns-0.awsdns-00.com."  
                },  
                {  
                    "Value": "ns-1024.awsdns-00.org."  
                },  
                {  
                    "Value": "ns-512.awsdns-00.net."  
                }  
            ]  
        },  
        {  
            "Name": "tutorial.",  
            "Type": "SOA",  
            "TTL": 900,  
            "ResourceRecords": [  
                {  
                    "Value": "ns-1536.awsdns-00.co.uk."  
                },  
                {  
                    "Value": "ns-0.awsdns-00.com."  
                },  
                {  
                    "Value": "ns-1024.awsdns-00.org."  
                },  
                {  
                    "Value": "ns-512.awsdns-00.net."  
                }  
            ]  
        }  
    ]  
}
```

```
        "Value": "ns-1536.awsdns-00.co.uk. awsdns-hostmaster.amazon.com. 1  
7200 900 1209600 86400"  
    }  
}  
,  
{  
    "Name": "myapplication.tutorial.",  
    "Type": "A",  
    "SetIdentifier": "16becc26-8558-4af1-9fdb-f81be062a266",  
    "MultiValueAnswer": true,  
    "TTL": 300,  
    "ResourceRecords": [  
        {  
            "Value": "172.31.87.2"  
        }  
    ]  
}  
]  
}
```

5. You can also query the DNS using dig from an instance within your VPC with the following command:

```
dig +short myapplication.tutorial
```

Output:

```
172.31.87.2
```

Step 4: Clean up

When you are finished with this tutorial, you should clean up the associated resources to avoid incurring charges for unused resources.

To clean up the service discovery instances and Amazon ECS resources

1. Deregister the service discovery service instances:

```
aws servicediscovery deregister-instance --service-id srv-utcrh6wavdkggqtk --instance-id 16becc26-8558-4af1-9fdb-f81be062a266 --region us-east-1
```

Output:

```
{  
    "OperationId": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv"  
}
```

2. Using the OperationId from the previous output, verify that the service discovery service instances were deregistered successfully:

```
aws servicediscovery get-operation --operation-id xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv --region us-east-1
```

Output:

```
{
```

```

    "Operation": {
        "Id": "xhu73bsertlyffhm3faqi7kumsmx274n-jh0zimzv",
        "Type": "Deregister_Instance",
        "Status": "Success",
        "CreateDate": 1525984073.707,
        "UpdateDate": 1525984076.426,
        "Targets": [
            {
                "INSTANCE": "16becc26-8558-4af1-9fbd-f81be062a266",
                "ROUTE_53_CHANGE_ID": "C5NSRG1J4I1FH",
                "SERVICE": "srv-utcrh6wavdkggqtk"
            }
        ]
    }
}

```

3. Delete the service discovery service:

```
aws servicediscovery delete-service --id srv-utcrh6wavdkggqtk --region us-east-1
```

4. Delete the service discovery namespace:

```
aws servicediscovery delete-namespace --id ns-uejictsjen2i4eeg --region us-east-1
```

Output:

```
{
    "OperationId": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj"
}
```

5. Using the OperationId from the previous output, verify that the service discovery namespace was deleted successfully:

```
aws servicediscovery get-operation --operation-id c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj --region us-east-1
```

Output:

```
{
    "Operation": {
        "Id": "c3ncqglftesw4ibgj5baz6ktaoh6cg4t-jh0ztysj",
        "Type": "Delete_Namespace",
        "Status": "Success",
        "CreateDate": 1525984602.211,
        "UpdateDate": 1525984602.558,
        "Targets": [
            {
                "NAMESPACE": "ns-rymlehshst7hhukh",
                "ROUTE_53_CHANGE_ID": "CJP2A2M86XW3O"
            }
        ]
    }
}
```

6. Update the Amazon ECS service so that the desired count is 0, which allows you to delete it:

```
aws ecs update-service --cluster tutorial --service ecs-service-discovery --desired-count 0 --force-new-deployment --region us-east-1
```

7. Delete the Amazon ECS service:

```
aws ecs delete-service --cluster tutorial --service ecs-service-discovery --region us-east-1
```

8. Delete the Amazon ECS cluster:

```
aws ecs delete-cluster --cluster tutorial --region us-east-1
```

Tutorial: Creating a service using a blue/green deployment

Amazon ECS has integrated blue/green deployments into the Create Service wizard on the Amazon ECS console. For more information, see [Creating an Amazon ECS service \(p. 546\)](#).

The following tutorial shows how to create an Amazon ECS service containing a Fargate task that uses the blue/green deployment type with the AWS CLI.

Note

Support for performing a blue/green deployment has been added for AWS CloudFormation. For more information, see [Perform Amazon ECS blue/green deployments through CodeDeploy using AWS CloudFormation](#) in the *AWS CloudFormation User Guide*.

Prerequisites

This tutorial assumes that you have completed the following prerequisites:

- The latest version of the AWS CLI is installed and configured. For more information about installing or upgrading the AWS CLI, see [Installing the AWS Command Line Interface](#).
- The steps in [Setting up with Amazon ECS \(p. 7\)](#) have been completed.
- Your AWS user has the required permissions specified in the [Amazon ECS first-run wizard permissions \(p. 664\)](#) IAM policy example.
- You have a VPC and security group created to use. For more information, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters](#).
- The Amazon ECS CodeDeploy IAM role is created. For more information, see [Amazon ECS CodeDeploy IAM Role \(p. 704\)](#).

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 \(p. 691\)](#).

```
{
    "family": "tutorial-task-def",
    "networkMode": "awsvpc",
    "containerDefinitions": [
        {
            "name": "sample-app",
            "image": "httpd:2.4",
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "entryPoint": [
                "sh",
                "-c"
            ],
            "command": [
                "/bin/sh -c \"echo '<html> <head> <title>Amazon ECS Sample App</title>
<style>body {margin-top: 40px; background-color: #333;} </style> </head><body> <div
style=color:white;text-align:center> <h1>Amazon ECS Sample App</h1> <h2>Congratulations!
</h2> <p>Your application is now running on a container in Amazon ECS.</p> </div></body></
html>' > /usr/local/apache2/htdocs/index.html && httpd-foreground\""
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512",
    "executionRoleArn": "arn:aws:iam::aws_account_id:role/ecstaskExecutionRole"
}
```

Then register the task definition using the `fargate-task.json` file that you created.

```
aws ecs register-task-definition \
--cli-input-json file://fargate-task.json \
--region us-east-1
```

Step 4: Create an Amazon ECS service

Use the [create-service](#) command to create a service.

First, create a file named `service-bluegreen.json` with the following contents.

```
{
    "cluster": "tutorial-bluegreen-cluster",
    "serviceName": "service-bluegreen",
```

```

"taskDefinition": "tutorial-task-def",
"loadBalancers": [
    {
        "targetGroupArn":
"arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4",
        "containerName": "sample-app",
        "containerPort": 80
    }
],
"launchType": "FARGATE",
"schedulingStrategy": "REPLICA",
"deploymentController": {
    "type": "CODE_DEPLOY"
},
"platformVersion": "LATEST",
"networkConfiguration": {
    "awsvpcConfiguration": {
        "assignPublicIp": "ENABLED",
        "securityGroups": [ "sg-abcd1234" ],
        "subnets": [ "subnet-abcd1234", "subnet-abcd5678" ]
    }
},
"desiredCount": 1
}
}

```

Then create your service using the `service-bluegreen.json` file that you created.

```

aws ecs create-service \
--cli-input-json file://service-bluegreen.json \
--region us-east-1

```

The output includes the ARN of the service, with the following format:

```
arn:aws:ecs:region:aws_account_id:service/service-bluegreen
```

Step 5: Create the AWS CodeDeploy resources

Use the following steps to create your CodeDeploy application, the Application Load Balancer target group for the CodeDeploy deployment group, and the CodeDeploy deployment group.

To create CodeDeploy resources

1. Use the [create-application](#) command to create 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 \(p. 704\)](#).

```
{
    "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": [
        {
            "identifier": "ecs-svc/9223370493425779968",
            "desiredCount": 1,
            "pendingCount": 0,
            "runningCount": 1,
            "status": "ACTIVE",
            "trafficWeight": 0.0,
            "targetGroup": {
                "name": "bluegreentarget1"
            }
        },
        {
            "identifier": "ecs-svc/9223370493423413672",
            "desiredCount": 1,
            "pendingCount": 0,
            "runningCount": 1,
            "status": "PRIMARY",
            "trafficWeight": 100.0,
            "targetGroup": {
                "name": "bluegreentarget2"
            }
        }
    ]
}
```

Step 7: Clean up

When you have finished this tutorial, clean up the resources associated with it to avoid incurring charges for resources that you aren't using.

Cleaning up the tutorial resources

1. Use the [delete-deployment-group](#) command to delete the CodeDeploy deployment group.

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

2. Use the [delete-application](#) command to delete the CodeDeploy application.

```
aws deploy delete-application \
--application-name tutorial-bluegreen-app \
--region us-east-1
```

3. Use the [delete-service](#) command to delete the Amazon ECS service. Using the --force flag allows you to delete a service even if it has not been scaled down to zero tasks.

```
aws ecs delete-service \
--service arn:aws:ecs:region:aws_account_id:service/service-bluegreen \
--force \
--region us-east-1
```

4. Use the [delete-cluster](#) command to delete the Amazon ECS cluster.

```
aws ecs delete-cluster \
--cluster tutorial-bluegreen-cluster \
--region us-east-1
```

5. Use the [s3 rm](#) command to delete the AppSpec file from the Amazon S3 bucket.

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

6. Use the [s3 rb](#) command to delete the Amazon S3 bucket.

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

7. Use the [delete-load-balancer](#) command to delete the Application Load Balancer.

```
aws elbv2 delete-load-balancer \
--load-balancer-arn
arn:aws:elasticloadbalancing:region:aws_account_id:loadbalancer/app/bluegreen-alb/
e5ba62739c16e642 \
--region us-east-1
```

8. Use the [delete-target-group](#) command to delete the two Application Load Balancer target groups.

```
aws elbv2 delete-target-group \
--target-group-arn
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget1/209a844cd01825a4 \
--region us-east-1
```

```
aws elbv2 delete-target-group \
--target-group-arn
arn:aws:elasticloadbalancing:region:aws_account_id:targetgroup/
bluegreentarget2/708d384187a3cfdc \
--region us-east-1
```

Tutorial: Listening for Amazon ECS CloudWatch Events

In this tutorial, you set up a simple AWS Lambda function that listens for Amazon ECS task events and writes them out to a CloudWatch Logs log stream.

Prerequisite: Set up a test cluster

If you do not have a running cluster to capture events from, follow the steps in [Creating a cluster \(p. 176\)](#) to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Lambda function correctly.

Step 1: Create the Lambda function

In this procedure, you create a simple Lambda function to serve as a target for Amazon ECS event stream messages.

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose **Create function**.
3. On the **Author from scratch** screen, do the following:
 - a. For **Name**, enter a value.
 - b. For **Runtime**, choose **Python 2.7**.
 - c. For **Role**, choose **Create a new role with basic Lambda permissions**.
4. Choose **Create function**.
5. In the **Function code** section, edit the sample code to match the following example:

```
import json

def lambda_handler(event, context):
    if event["source"] != "aws.ecs":
        raise ValueError("Function only supports input from events with a source type of: aws.ecs")

    print('Here is the event:')
    print(json.dumps(event))
```

This is a simple Python 2.7 function that prints the event sent by Amazon ECS. If everything is configured correctly, at the end of this tutorial, you see that the event details appear in the CloudWatch Logs log stream associated with this Lambda function.

6. Choose **Save**.

Step 2: Register 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 a CloudWatch Events event rule that only captures task events where the task has stopped running because one of its essential containers has terminated. The event sends only task events with a specific `stoppedReason` property to the designated Amazon SNS topic.

Prerequisite: Set up a test cluster

If you do not have a running cluster to capture events from, follow the steps in [Creating a cluster \(p. 176\)](#) to create one. At the end of this tutorial, you run a task on this cluster to test that you have configured your Amazon SNS topic and CloudWatch Events event rule correctly.

Step 1: Create and subscribe to an Amazon SNS topic

For this tutorial, you configure an Amazon SNS topic to serve as an event target for your new event rule.

To create an Amazon SNS topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. Choose **Topics, Create topic**.
3. On the **Create topic** screen, for **Name**, enter **TaskStoppedAlert** and choose **Create topic**.
4. On the **TaskStoppedAlert** details screen, choose **Create subscription**.
5. On the **Create subscription** screen, for **Protocol**, choose **Email**. For **Endpoint**, enter an email address to which you currently have access and choose **Create subscription**.
6. Check your email account, and wait to receive a subscription confirmation email message. When you receive it, choose **Confirm subscription**.

Step 2: Register an event rule

Next, you register an event rule that captures only task-stopped events for tasks with stopped containers.

To create an event rule

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. On the navigation pane, choose **Events, Rules, Create rule**.
3. For Event Source, choose **Event Pattern**, select **Custom event pattern** and then replace the existing text with the following text:

```
{  
  "source": [  
    "aws.ecs"  
  ],  
  "detail-type": [  
    "ECS Task State Change"  
  ],  
  "detail": {  
    "lastStatus": [  
      "STOPPED"  
    ],  
    "stoppedReason": [  
      "Essential container in task exited"  
    ]  
  }  
}
```

This code defines a CloudWatch Events event rule that matches any event where the `lastStatus` and `stoppedReason` fields match the indicated values. For more information about event patterns, see [Events and Event Patterns](#) in the *Amazon CloudWatch User Guide*.

4. For **Targets**, choose **Add target**. For **Target type**, choose **SNS topic**, and then choose **TaskStoppedAlert**.
5. Choose **Configure details**.
6. For **Rule definition**, type a name and description for your rule and then choose **Create rule**.

Step 3: Test your rule

Verify that the rule is working by running a task that exits shortly after it starts. If your event rule is configured correctly, you receive an email message within a few minutes with the event text. If you have

an existing task definition that can satisfy the rule requirements, run a task using it. If you do not, the following steps will walk you through registering a Fargate task definition and running it that will.

To test the rule

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. Choose **Task Definitions, Create new Task Definition**.
3. For Select launch type compatibility, choose **FARGATE**, **Next step**.
4. Choose **Configure via JSON**, copy and paste the following task definition JSON into the field and choose **Save**.

```
{  
    "containerDefinitions": [  
        {  
            "command": [  
                "sh",  
                "-c",  
                "sleep 5"  
            ],  
            "essential": true,  
            "image": "amazonlinux:2",  
            "name": "test-sleep"  
        }  
    ],  
    "cpu": "256",  
    "executionRoleArn": "arn:aws:iam::012345678910:role/ecsTaskExecutionRole",  
    "family": "fargate-task-definition",  
    "memory": "512",  
    "networkMode": "awsvpc",  
    "requiresCompatibilities": [  
        "FARGATE"  
    ]  
}
```

5. Choose **Create, View task definition**.
6. For **Actions**, choose **Run Task**.
7. For Launch type, choose **FARGATE**. For **VPC and security groups**, choose a VPC and Subnets for the task to use and then choose **Run Task**.
8. For **Container name**, type **Wordpress**, for **Image**, type **wordpress**, and for **Maximum memory (MB)**, type **128**.
9. On the **Tasks** tab for your cluster, periodically choose the refresh icon until you no longer see your task running. To verify that your task has stopped, for **Desired task status**, choose **Stopped**.
10. Check your email to confirm that you have received an email alert for the stopped notification.

Tutorial: Using Amazon EFS file systems with Amazon ECS

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

You can use Amazon EFS file systems with Amazon ECS to access file system data across your fleet of Amazon ECS tasks. That way, your tasks have access to the same persistent storage, no matter the infrastructure or container instance on which they land. When you reference your Amazon EFS file system and container mount point in your Amazon ECS task definition, Amazon ECS takes care of

mounting the file system in your container. The following sections help you get started using Amazon EFS with Amazon ECS.

This feature is supported by tasks that use both the EC2 and Fargate launch types, however this tutorial will use an Amazon ECS task that uses the EC2 launch type. This tutorial is also meant to be followed step by step, however if you already have some of these resources created on your account then you may be able to skip some steps.

Note

Amazon EFS may not be available in all Regions. For more information about which Regions support Amazon EFS, see [Amazon Elastic File System Endpoints and Quotas](#) in the *AWS General Reference*.

Step 1: Create an Amazon ECS cluster

Use the following steps to create an Amazon ECS cluster. When you use the AWS Management Console to create a non-empty cluster, Amazon ECS creates an AWS CloudFormation stack along with Auto Scaling resources.

To create an Amazon ECS cluster

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the navigation bar at the top of the screen, select the **US West (Oregon)** Region.
3. In the navigation pane, choose **Clusters**.
4. On the **Clusters** page, choose **Create Cluster**.
5. For **Select cluster compatibility**, choose **EC2 Linux + Networking** and then choose **Next step**.
6. For **Cluster name**, enter `EFS-tutorial` for the cluster name.
7. For **Provisioning model**, choose **On-Demand Instance**.
8. For **EC2 instance type**, choose `t2.micro`.
9. For **Number of instances**, enter `1`.
10. For **EC2 AMI Id**, choose the Amazon Linux 2 Amazon ECS-optimized AMI.
11. For **EBS storage (GiB)**, leave the default setting.
12. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for SSH access. This is required as you will connect to the instance later.
13. In the **Networking** section, configure the VPC to launch your container instances into. By default, the cluster creation wizard creates a new VPC with two subnets in different Availability Zones, and a security group open to the internet on port 80. This is a basic setup that works well for an HTTP service. However, you can modify these settings by following the steps below.

Important

Record the VPC and security group IDs you use for your cluster as you will need to create the Amazon EFS file system in the same VPC.

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

You will also use this security group when you create a security group for the Amazon EFS file system, so note the ID.

Note

You can also choose to create a new security group and then modify the rules after the cluster is created. For more information, see [Amazon EC2 Security Groups for Linux Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

- e. In the **Container instance IAM role** section, select the IAM role to use with your container instances. If your account has the **ecsInstanceRole** that is created for you in the console first-run wizard, it is selected by default. If you do not have this role in your account, you can choose to create the role, or you can choose another IAM role to use with your container instances.

Important

The IAM role you use must have the `AmazonEC2ContainerServiceforEC2Role` managed policy attached to it, otherwise you will receive an error during cluster creation. If you do not launch your container instance with the proper IAM permissions, your Amazon ECS agent does not connect to your cluster. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

- f. For **CloudWatch Container Insights**, deselect **Enable Container Insights** as this feature won't be needed for this tutorial.
- g. Choose **Create**.

Step 2: Create a security group for the Amazon EFS file system

In this step, you create a security group for your Amazon EFS file system that allows inbound access from your container instances. This security group will contain an inbound rule that references the security group you created, or referenced, for your cluster in the previous step.

To create a security group for an Amazon EFS file system

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the left navigation pane, choose **Security Groups**, **Create security group**.
3. For **Security group name**, enter a unique name for your security group. For example, `EFS-access-for-sg-dc025fa2`.
4. For **Description**, enter a description for your security group.
5. For **VPC**, choose the VPC that you identified earlier for your cluster.
6. For **Inbound rules**, choose **Add rule**.
7. For **Type**, choose **NFS**.
8. For **Source**, choose **Custom** and then enter the security group ID that you identified when you created the cluster.
9. Choose **Create security group**.

Step 3: Create an Amazon EFS file system

In this step, you create an Amazon EFS file system.

To create an Amazon EFS file system for Amazon ECS tasks.

1. Open the Amazon Elastic File System console at <https://console.aws.amazon.com/efs/>.

2. Choose **Create file system**.
 3. On the **Configure network access** page, choose the VPC that your container instances are hosted in. By default, each subnet in the specified VPC receives a mount target that uses the default security group for that VPC.
- Important**
Your Amazon EFS file system, your Amazon ECS cluster, container instances and tasks must be in the same VPC.
4. Under **Create mount targets**, for **Security groups**, add the security group that you created in step 2. Choose **Next Step**.
 5. On the **Configure file system settings** page, configure optional settings and then choose **Next Step** to proceed.
 - a. (Optional) Add tags for your file system. For example, you could specify a unique name for the file system by entering that name in the **Value** column next to the **Name** key.
 - b. (Optional) Enable lifecycle management to save money on infrequently accessed storage. For more information, see [EFS Lifecycle Management](#) in the *Amazon Elastic File System User Guide*.
 - c. Choose a throughput mode for your file system. The **Bursting** mode is the default, and it is recommended for most file systems.
 - d. Choose a performance mode for your file system. The **General Purpose** mode is the default, and it is recommended for most file systems.
 - e. (Optional) Enable encryption. Select the check box to enable encryption of your Amazon EFS file system at rest.
 6. On the **Configure client access** page, choose **Next Step**.
 7. Review your file system options and choose **Create File System** to complete the process.
 8. From the file systems details screen, record the **File system ID**. In the next step, you will reference this value in your Amazon ECS task definition.

Step 4: Add content to the Amazon EFS file system

In this step, you mount the Amazon EFS file system to an Amazon EC2 instance and add content to it. This is for testing purposes in this tutorial, to illustrate the persistent nature of the data. When using this feature you would normally have your application or another method of writing data to your Amazon EFS file system.

To create an Amazon EC2 instance and mount the Amazon EFS file system

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Launch Instance**.
3. On the **Choose an Amazon Machine Image** page, select the latest **Amazon Linux 2 AMI (HVM)** AMI.
4. On the **Choose an Instance Type** page, keep the default instance type, **t2.micro** and choose **Next: Configure Instance Details**.
5. On the **Configure Instance Details** page, do the following:
 - a. For **Network**, select the VPC that you specified for your Amazon EFS file system and Amazon ECS cluster.
 - b. For **Auto-assign Public IP**, choose **Enable**. Otherwise, your instances do not get public IP addresses or public DNS names.
 - c. For **File systems**, select your Amazon EFS file system. You can optionally change the mount location or leave the default value.
 - d. Under **Advanced Details**, ensure that the user data script is populated automatically with the Amazon EFS file system mounting steps.

6. Advance to step 5 of the instance wizard by choosing **Next: Add Storage**, **Next: Add Tags**, and **Next: Configure Security Group**.
7. On the **Configure Security Group** page, choose **Select an existing security group** and select the security group that you created in step 1, and then choose **Review and Launch**.
8. On the **Review Instance Launch** page, choose **Launch**.
9. On the **Select an existing key pair or create a new key pair** dialog box, select **Choose an existing key pair** and choose your key pair. Select the acknowledgment check box, and choose **Launch Instances**.
10. On the **Launch Status** page, choose **View Instances** to see the status of your instances. Initially, their status is pending. After the status changes to **running**, your instances are ready for use.

Now, you connect to the Amazon EC2 instance and add content to the Amazon EFS file system.

To connect to the Amazon EC2 instance and add content to the Amazon EFS file system

1. SSH to the Amazon EC2 instance you created. For more information, see [Connect to Your Linux Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.
2. From the terminal window for each instance, run the **df -T** command to verify that the Amazon EFS file system is mounted. In the following output, we have highlighted the Amazon EFS file system mount.

```
$ df -T
Filesystem      Type      1K-blocks    Used   Available Use% Mounted on
devtmpfs        devtmpfs    485468      0       485468  0% /dev
tmpfs           tmpfs      503480      0       503480  0% /dev/shm
tmpfs           tmpfs      503480     424     503056  1% /run
tmpfs           tmpfs      503480      0       503480  0% /sys/fs/cgroup
/dev/xvda1      xfs       8376300  1310952    7065348 16% /
127.0.0.1:/    nfs4      9007199254739968    0  9007199254739968  0% /mnt/efs/fs1
tmpfs           tmpfs      100700      0       100700  0% /run/user/1000
```

3. Navigate to the directory that the Amazon EFS file system is mounted at. In the example above, that is `/mnt/efs/fs1`.
4. Create a file named `index.html` with the following content:

```
<html>
  <body>
    <h1>It Works!</h1>
    <p>You are using an Amazon EFS file system for persistent container storage.</p>
  </body>
</html>
```

Step 5: Create a task definition

The following task definition creates a data volume named `efs-html`. The `nginx` container mounts the host data volume at the NGINX root, `/usr/share/nginx/html`.

To create a new task definition

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. In the navigation pane, choose **Task Definitions**, **Create new Task Definition**.
3. On the **Select compatibilities** page, choose **EC2**, **Next step**.

4. Choose **Configure via JSON**, copy and paste the following JSON text, replacing the `fileSystemId` with the ID of your Amazon EFS file system.

```
{  
    "containerDefinitions": [  
        {  
            "memory": 128,  
            "portMappings": [  
                {  
                    "hostPort": 80,  
                    "containerPort": 80,  
                    "protocol": "tcp"  
                }  
            ],  
            "essential": true,  
            "mountPoints": [  
                {  
                    "containerPath": "/usr/share/nginx/html",  
                    "sourceVolume": "efs-html"  
                }  
            ],  
            "name": "nginx",  
            "image": "nginx"  
        }  
    ],  
    "volumes": [  
        {  
            "name": "efs-html",  
            "efsVolumeConfiguration": {  
                "fileSystemId": "fs-1324abcd",  
                "transitEncryption": "ENABLED"  
            }  
        }  
    ],  
    "family": "efs-tutorial"  
}
```

5. Choose **Save, Create**.

Step 6: Run a task and view the results

Now that your Amazon EFS file system is created and there is web content for the NGINX container to serve, you can run a task using the task definition that you created. The NGINX web server serves your simple HTML page. If you update the content in your Amazon EFS file system, those changes are propagated to any containers that have also mounted that file system.

To run a task and view the results

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. Choose the cluster that you created in step 1 earlier.
3. Choose **Tasks, Run new task**.
4. For **Task Definition**, choose the `nginx-efs` task definition that you created earlier and choose **Run Task**. For more information on the other options in the run task workflow, see [Run a standalone task \(p. 510\)](#).
5. Below the **Tasks** tab, choose the task that you just ran.
6. Expand the container name at the bottom of the page, and choose the IP address that is associated with the container. Your browser should open a new tab with the following message:

It Works!

You are using an Amazon EFS file system for persistent container storage.

Note

If you do not see the message, make sure that the security group for your container instance allows inbound network traffic on port 80.

Tutorial: Using FSx for Windows File Server file systems with Amazon ECS

FSx for Windows File Server provides fully managed Microsoft Windows file servers, that are backed by a fully native Windows file system. When using FSx for Windows File Server together with ECS, you can provision your Windows tasks with persistent, distributed, shared, static file storage. For more information, see [What Is FSx for Windows File Server?](#) in the *FSx for Windows File Server User Guide*.

You can use FSx for Windows File Server to deploy Windows workloads that require access to shared external storage, highly available regional storage, or high-throughput storage. You can mount one or more FSx for Windows File Server file system volumes to an ECS container running on an ECS Windows instance. You can share FSx for Windows File Server file system volumes among multiple ECS containers within a single ECS task.

Note

FSx for Windows File Server might not be available in all Regions. For more information about which Regions support FSx for Windows File Server, see [Amazon FSx Endpoints and Quotas](#) in the *AWS General Reference*.

In this tutorial, you launch an ECS Optimized Windows instance that hosts an FSx for Windows File Server file system and containers that can access the file system. To do this, you first create an AWS Directory Service AWS Managed Microsoft Active Directory. Then, you create an Amazon FSx for Windows File Server file system and an ECS cluster with an ECS instance and an ECS task definition. You configure the task definition for your containers to use the FSx for Windows File Server file system. Finally, you test the file system.

It takes 20 to 45 minutes each time you launch or delete either the Active Directory or the FSx for Windows File Server file system. Be prepared to reserve at least 90 minutes to complete the tutorial or complete the tutorial over a few sessions.

Prerequisites for the tutorial

- An IAM Account with administrator access. See [Setting up with Amazon ECS \(p. 7\)](#).
- (Optional) A pem key pair for connecting to your EC2 Windows instance through RDP access. For information about how to create key pairs, see [Amazon EC2 key pairs and Windows instances](#) in the *User Guide for Windows Instances*.
- A VPC with at least one public and one private subnet, and one security group. You can use your default VPC. You don't need a NAT gateway or device. AWS Directory Service doesn't support Network Address Translation (NAT) with Active Directory. For this to work, the Active Directory, FSx for Windows File Server file system, ECS Cluster, and ECS instance must be located within your VPC. For more

information regarding VPCs and Active Directories, see [Amazon VPC console wizard configurations](#) and [AWS Managed Microsoft AD Prerequisites](#).

- The IAM ecsInstanceRole and ecsTaskExecutionRole permissions are associated with your account. These service-linked roles allow services to make API calls and access containers, secrets, directories and file servers on your behalf.

Step 1: Create IAM access roles

Create a cluster with the AWS Management Console.

1. See [Amazon ECS container instance IAM role \(p. 695\)](#) to check whether you have an ecsInstanceRole and to see how you can create one if you don't have one.
2. We recommend that role policies are customized for minimum permissions in an actual production environment. For the purpose of working through this tutorial, verify that the following AWS managed policy is attached to your ecsInstanceRole. Attach the policy if it is not already attached.
 - AmazonEC2ContainerServiceforEC2Role

To attach AWS managed policies.

- a. Open the [IAM console](#).
 - b. In the navigation pane, choose **Roles**.
 - c. Choose an [AWS managed role](#).
 - d. Choose **Permissions, Attach policies..**.
 - e. To narrow the available policies to attach, use **Filter**.
 - f. Select the appropriate policy and choose **Attach policy**.
3. See [Amazon ECS task execution IAM role \(p. 691\)](#) to check whether you have an ecsTaskExecutionRole and to see how you can create one if you don't have one.

We recommend that role policies are customized for minimum permissions in an actual production environment. For the purpose of working through this tutorial, verify that the following AWS managed policies are attached to your ecsTaskExecutionRole. Attach the policies if they are not already attached. Use the procedure given in the preceding section to attach the AWS managed policies.

- SecretsManagerReadWrite
- AmazonFSxReadOnlyAccess
- AmazonSSMReadOnlyAccess
- AmazonECSTaskExecutionRolePolicy

Step 2: Create Windows Active Directory (AD)

1. Follow the steps described in [Create Your AWS Managed AD Directory](#) in the *AWS Directory Service Administration Guide*. Use the VPC you have designated for this tutorial. On Step 3 of *Create Your AWS Managed AD Directory*, save the user name and password for use in a following step. Also, note the fully qualified domain name for future steps. You can go on to complete the following step while the Active Directory is being created.
2. Create an AWS Secrets Manager secret to use in the following steps. For more information, see [Getting Started with AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.
 - a. Open the [Secrets Manager console](#).

- b. Click **Store a new secret**.
- c. Select **Other type of secrets**.
- d. For **Secret key/value**, in the first row, create a key **username** with value **admin**. Click on **+ Add row**.
- e. In the new row, create a key **password**. For value, type in the password you entered in Step 3 of *Create Your AWS Managed AD Directory*.
- f. Click on the **Next** button.
- g. Provide a secret name and description. Click **Next**.
- h. Click **Next**. Click **Store**.
- i. From the list of **Secrets** page, click on the secret you have just created.
- j. Save the ARN of the new secret for use in the following steps.
- k. You can proceed to the next step while your Active Directory is being created.

Step 3: Verify and update your security group

In this step, you verify and update the rules for the security group that you're using. For this, you can use the default security group that was created for your VPC.

Verify and update security group.

You need to create or edit your security group to send data from and to the ports, which are described in [Amazon VPC Security Groups](#) in the *FSx for Windows File Server User Guide*. You can do this by creating the security group inbound rule shown in the first row of the following table of inbound rules. This rule allows inbound traffic from network interfaces (and their associated instances) that are assigned to the security group. All of the cloud resources you create are within the same VPC and attached to the same security group. Therefore, this rule allows traffic to be sent to and from the FSx for Windows File Server file system, Active Directory, and ECS instance as required. The other inbound rules allow traffic to serve the website and RDP access for connecting to your ECS instance.

The following table shows which security group inbound rules are required for this tutorial.

Type	Protocol	Port range	Source
All traffic	All	All	<i>sg-securitygroup</i>
Custom TCP	TCP	8080	0.0.0.0/0
RDP	TCP	3389	your EC2 instance public IP address

The following table shows which security group outbound rules are required for this tutorial.

Type	Protocol	Port range	Destination
All traffic	All	All	0.0.0.0/0

1. Open the [EC2 console](#) and select **Security Groups** from the left-hand menu.
2. From the list of security groups now displayed, select check the check-box to the left of the security group that you are using for this tutorial.

Your security group details are displayed.

3. Edit the inbound and outbound rules by selecting the **Inbound rules** or **Outbound rules** tabs and choosing the **Edit inbound rules** or **Edit outbound rules** buttons. Edit the rules to match those displayed in the preceding tables. After you create your EC2 instance later on in this tutorial, edit the inbound rule RDP source with the public IP address of your EC2 instance as described in [Connect to your Windows instance](#) from the *Amazon EC2 User Guide for Windows Instances*.

Step 4: Create an FSx for Windows File Server file system

After your security group is verified and updated and your Active Directory is created and is in the active status, create the FSx for Windows File Server file system in the same VPC as your Active Directory. Use the following steps to create an FSx for Windows File Server file system for your Windows tasks.

Create your first file system.

1. Open the [Amazon FSx console](#).
2. On the dashboard, choose **Create file system** to start the file system creation wizard.
3. On the **Select file system type** page, choose **FSx for Windows File Server**, and then choose **Next**. The **Create file system** page appears.
4. In the **File system details** section, provide a name for your file system. Naming your file systems makes it easier to find and manage them. You can use up to 256 Unicode characters. Allowed characters are letters, numbers, spaces, and the special characters plus sign (+), minus sign (-), equal sign (=), period (.), underscore (_), colon (:), and forward slash (/).
5. For **Deployment type** choose **Single-AZ** to deploy a file system that is deployed in a single Availability Zone. *Single-AZ 2* is the latest generation of single Availability Zone file systems, and it supports SSD and HDD storage.
6. For **Storage type**, choose **HDD**.
7. For **Storage capacity**, enter the minimum storage capacity.
8. Keep **Throughput capacity** at its default setting.
9. In the **Network & security** section, choose the same Amazon VPC that you chose for your AWS Directory Service directory.
10. For **VPC Security Groups**, choose the security group that you verified in *Step 3: Verify and update your security group*.
11. For **Windows authentication**, choose **AWS Managed Microsoft Active Directory**, and then choose your AWS Directory Service directory from the list.
12. For **Encryption**, keep the default **Encryption key** setting of **aws/fsx (default)**.
13. Keep the default settings for **Maintenance preferences**.
14. Click on the **Next** button.
15. Review the file system configuration shown on the **Create file system** page. For your reference, note which file system settings you can modify after file system is created. Choose **Create file system**.
16. Note the file system ID. You will need to use it in a later step.

You can go on to the next steps to create a cluster and EC2 instance while the FSx for Windows File Server file system is being created.

Step 5: Create an Amazon ECS cluster

Create a cluster using the AWS Management Console.

1. Open the [Amazon ECS console](#).

2. In the navigation pane, choose **Clusters**.
3. On the **Clusters** page, choose **Create Cluster**.
4. Choose **EC2 Windows + Networking** and choose **Next step**.
5. For **Cluster name** enter `windows-fsx-cluster`.
6. Click the check-box under the name of the cluster to **create an empty cluster**.
7. Click the **Create** button on the lower right corner.
8. Click the **View Cluster** button when the cluster is successfully created.

You are now on a page where you can view the details of your cluster.

Step 6: Create an Amazon ECS instance

Launch an ECS Optimized Windows EC2 instance into the ECS cluster you just created using the AWS Management Console.

1. Go to [Amazon ECS-optimized AMI](#) in the *Amazon ECS Developer Guide* to find the latest version of the Windows Server 2019 Full AMI in the same Region as your VPC.
2. You can get the latest version using one of the following steps.

Scroll down to the Windows Server 2019 Full AMI table.

- a. Find the latest version in the table for your Region. Click **View AMI ID** link to a page where you'll find the AMI ID of the latest version. Save a copy of the AMI ID for the next steps.
- b. Run the given Systems Manager command using the AWS CLI and save a copy of the AMI ID that is returned.

3. Open the [Amazon EC2 console](#).
4. Click on the **Launch Instance** button and select **Launch Instance**.

You are now on a page that lists available EC2 instances.

5. **Select an AMI for your EC2 instance.**
 - a. Under **Quick Start**, click on **Community AMIs**.
 - b. In the **search** field, enter the AMI ID that you saved from the previous step and press return.
 - c. Select the Windows Server 2019 Full AMI that matches the AMI ID that you saved in the previous step.

You are now on a page listing instance types.

6. For **Instance type** page, choose t2.medium or t2.micro and click on **Next: Configure Instance Details**.
7. **Configure instance details.**
 - a. On the **Configure Instance Details** page, enter 1 for **Number of Instances**.
 - b. For **Network** select your VPC.
 - c. For **Subnets** select a public subnet.
 - d. Select **Enable** for **Auto-assign Public IP**.
 - e. For **Domain join directory**, select the ID of the Active Directory that you created. This option domain joins your AD when the EC2 instance is launched.
 - f. For **IAM role**, select your **ecsInstanceRole** from the drop-down menu.
 - g. Scroll to the bottom of the page and enter the following into the **User data** text field.

```
<powershell>
```

```
Initialize-ECSAgent -Cluster windows-fsx-cluster -EnableTaskIAMRole
</powershell>
```

- h. Click **Next: Add Storage** button.
- i. Click **Next: Add Tags** button.
- j. Click **Next: Configure Security Group** button.
8. On the **Configure Security Group** page, select the security group that you verified and updated in *Step 3: Verify and update your security group*. If it doesn't already exist, add an inbound RDP TCP rule to allow traffic from your EC2 instance IP address through port 3389 if you want to be able to RDP into your instance.
9. Click on **Review and Launch** button.
10. On the **Review Instance Launch** page, click the **Launch** button.
11. For **Key pair**, choose an Amazon EC2 key pair to use with your container instances for RDP access. If you don't specify a key pair, you can't connect to your container instances with RDP. For more information, see [Prerequisites for the tutorial \(p. 796\)](#).
12. Click on **View instance** to see the new instance status among your list of instances.
13. Open the [Amazon ECS console](#) and select **Clusters**.
14. Select your **fsx-windows-cluster** cluster.
15. Select the **ECS Instances** tab and verify that your ECS instance has been registered in the **fsx-windows-cluster** cluster.

Step 7: Register a Windows task definition

Before you can run Windows containers in your Amazon ECS cluster, you must register a task definition. The following task definition example displays a simple web page on port 8080 of a container instance. The task launches two containers that have access to the FSx file system. The first container writes an HTML file to the file system. The second container downloads the HTML file from the file system and serves the webpage.

Register the sample task definition with the AWS Management Console.

1. Open the [Amazon ECS console](#).
2. In the navigation pane, choose **Task Definitions**.
3. On the **Task Definitions** page, choose **Create new Task Definition**.
4. On the **Select launch type compatibilities** page, choose **EC2** and then **Next step**.

Note

The Fargate launch type isn't compatible with Windows containers.

5. On the **Create new Task Definition** page, scroll to the bottom of the page and choose **Configure via JSON**.
6. Paste the following sample task definition JSON into the text area (replacing the pre-populated JSON there) and choose **Save**.

```
{
  "containerDefinitions": [
    {
      "entryPoint": [
        "powershell",
        "-Command"
      ],
      "portMappings": [],
      "command": [
        "New-Item -Path C:\\\\fsx-windows-dir\\\\index.html -ItemType file -Value '<html>
<head> <title>Amazon ECS Sample App</title> <style>body {margin-top: 40px; background-
```

```

color: #333; } </style> </head><body> <div style=color:white;text-align:center>
<h1>Amazon ECS Sample App</h1> <h2>It Works!</h2> <p>You are using Amazon FSx for
Windows File Server file system for persistent container storage.</p>' -Force"
],
"cpu": 512,
"memory": 256,
"image": "mcr.microsoft.com/windows/servercore/iis",
"essential": false,
"name": "container1"
},
{
"entryPoint": [
"powershell",
"-Command"
],
"portMappings": [
{
"hostPort": 8080,
"protocol": "tcp",
"containerPort": 80
}
],
"command": [
"Remove-Item -Recurse C:\\inetpub\\wwwroot\\* -Force; Start-Sleep -Seconds 120;
Move-Item -Path C:\\fsx-windows-dir\\index.html -Destination C:\\inetpub\\wwwroot\\
\\index.html -Force; C:\\ServiceMonitor.exe w3svc"
],
"cpu": 512,
"memory": 256,
"image": "mcr.microsoft.com/windows/servercore/iis",
"essential": true,
"name": "container2"
}
],
"family": "fsx-windows"
}

```

7. Directly above the **Configure via JSON** button, click the **plus sign** to the left of **Add volume**.
 - a. For **Volume type**, select **FSx for Windows File Server**.
 - b. For **Name**, enter **fsx-windows-vol** and save it for following steps.
 - c. For **File system ID**, select the ID of the FSx for Windows File Server file system that you created in preceding steps.
 - d. For **Root Directory**, enter **share**.
 - e. For **Credentials parameter**, enter the ARN of the secret you created by using the Secrets Manager in the preceding steps.
 - f. For **Domain**, enter your Active Directory fully qualified domain name.
8. Click on **container1** under **Container Definitions**.
9. Scroll to **STORAGE AND LOGGING** and, for **Mount points Source volume**, select **fsx-windows-vol** from the drop-down menu.
10. For **Container path**, enter **C:\\fsx-windows-dir**.
11. Click on **Update** button.
12. Repeat the last four steps for **container2** under **Container Definitions**.
13. For **Task execution role**, choose your **ecsTaskExecutionRole** from the drop-down menu.
14. Verify your information and click on **Create** button.

Step 8: Run a task and view the results

Before running the task, verify that the status of your FSx for Windows File Server file system is **Available**. After it is available, you can run a task using the task definition that you created. The task starts out by creating containers that shuffle an HTML file between them using the file system. After the shuffle, a web server serves the simple HTML page.

Note

You might not be able to connect to the website from within a VPN.

Run a task and view the results.

1. Open the [Amazon ECS console](#).
2. Choose your **fsx-windows-cluster** cluster.
3. Choose **Tasks** tab, and then **Run new task**.
4. For **Launch Type**, select EC2.
5. For **Task Definition**, choose the **fsx-windows** task definition that you created, and then choose **Run Task**.
6. Under the **Tasks** tab, choose the task that you just ran. Your task appears in the list of tasks.
7. When your task status is **RUNNING**, click on the task ID.
8. Expand container2.
9. Scroll down and click on the external IP address that is associated with the container. Your browser will open and display the following message.

Note

If you don't see this message, check that you aren't running in a VPN and make sure that the security group for your container instance allows inbound network HTTP traffic on port 8080.

Step 9: Clean up

Note

It takes 20 to 45 minutes to delete the FSx for Windows File Server file system or the AD. You must wait until the FSx for Windows File Server file system delete operations are complete before starting the AD delete operations.

Remove FSx for Windows File Server file system.

1. Open the [Amazon FSx console](#)
2. Click the radio button to the left of the FSx for Windows File Server file system that you just created.
3. Click on **Actions**.
4. Select **Delete file system**.

Remove AD.

1. Open the [AWS Directory Service console](#).
2. Click the radio button to the left of the AD you just created.
3. Click on **Actions**.
4. Select **Delete directory**.

Remove ECS cluster.

1. Open the [Amazon ECS console](#).

2. Select **Clusters**.
3. Select the cluster you just created.
4. Click the **Delete Cluster** button.

Remove ECS instance.

1. Open the [Amazon EC2 console](#).
2. From the left-hand menu, select **Instances**.
3. Check the check-box to the left of the EC2 instance you created during this exercise.
4. Click the **Instance state** and then **Terminate instance**.

Remove secret.

1. Open the [Secrets Manager console](#).
2. Select the secret you created for this walk through.
3. Click **Actions**.
4. Select **Delete secret**.

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 \(p. 805\)](#)
- [Troubleshooting ECS Anywhere issues \(p. 814\)](#)
- [Checking stopped tasks for errors \(p. 815\)](#)
- [CannotPullContainer task errors \(p. 817\)](#)
- [Service event messages \(p. 819\)](#)
- [Invalid CPU or memory value specified \(p. 824\)](#)
- [CannotCreateContainerError: API error \(500\): devmapper \(p. 825\)](#)
- [Troubleshooting service load balancers \(p. 826\)](#)
- [Troubleshooting service auto scaling \(p. 827\)](#)
- [Enabling Docker debug output \(p. 827\)](#)
- [Amazon ECS Log File Locations \(p. 828\)](#)
- [Amazon ECS logs collector \(p. 832\)](#)
- [Agent introspection diagnostics \(p. 833\)](#)
- [Docker diagnostics \(p. 835\)](#)
- [AWS Fargate throttling limits \(p. 837\)](#)
- [API failure reasons \(p. 837\)](#)
- [Troubleshooting IAM Roles for Tasks \(p. 839\)](#)

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 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 only supported for Linux containers 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 2004 Core AMI
 - Amazon ECS-optimized Windows Server 20H2 Core AMI
- ECS Exec is not currently supported using the AWS Management Console.
- 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 endpoints \(p. 716\)](#).
- 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 \(p. 813\)](#).
- 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.

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. 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": "ami-0a9a9a9a9a9a9a9a9"
    }
  ]
}
```

```
        "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:PutObjectAcl"  
            ],  
            "Resource": "arn:aws:s3:::s3-bucket-name/*"  
        }  
    ]  
}
```

```
        "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 inline policy to your task IAM role which requires the AWS KMS permissions. For more information, see [IAM permissions required for ECS Exec \(p. 807\)](#).

```
{
    "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",  
            "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*.

Limits 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

```

}

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 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 \(p. 807\)](#).
- The SSM Agent is not connected due to slowness or network latency. Wait and try the `execute-command` action again.

Troubleshooting ECS Anywhere issues

ECS Anywhere provides support for registering an *external instance* such as a on-premises server or virtual machine (VM) to your Amazon ECS cluster. The following are common issues that you might encounter and general troubleshooting recommendations for them.

Topics

- [External instance registration issues \(p. 814\)](#)
- [External instance network issues \(p. 815\)](#)
- [Issues running tasks on your external instance \(p. 815\)](#)

External instance registration issues

When registering an external instance with your Amazon ECS cluster, the following requirements must be met:

- An Systems Manager activation, which consists of an *activation ID* and *activation code*, must be retrieved. You use it to register the external instance as a Systems Manager managed instance. When a Systems Manager activation is requested, you can specify a registration limit and expiration date. The registration limit specifies the maximum number of instances that can be registered using the activation. For it, the default value is 1 instance. The expiration date specifies when the activation expires. The default value is 24 hours. If the Systems Manager activation that you're using to register your external instance isn't valid, request a new one. For more information, see [Registering an external instance to a cluster \(p. 418\)](#).
- An IAM policy is used to provide your external instance the permissions that it needs to communicate with AWS APIs. If this managed policy isn't created properly and doesn't contain the required permissions, external instance registration fails. For more information, see [Required IAM permissions for external instances \(p. 416\)](#).

- Amazon ECS provides an installation script that installs Docker, the Amazon ECS container agent, and the Systems Manager Agent on your external instance. If the installation script fails, it's likely that the script can't be run again on the same instance without an error occurring. If this happens, follow the cleanup process to clear AWS resources from the instance so you can run the installation script again. For more information, see [Deregistering an external instance \(p. 421\)](#).

Note

Be aware that, if the installation script successfully requested and used the Systems Manager activation, running the installation script a second time uses the Systems Manager activation again. This might in turn cause you to reach the registration limit for that activation. If this limit is reached, you must create a new activation.

- When running the installation script on an external instance for GPU workloads, if the NVIDIA driver is not detected or configured properly an error will occur. The installation script uses the `nvidia-smi` command to confirm the existence of the NVIDIA driver.

External instance network issues

To communicate any changes, your external instance requires a network connection to AWS. If your external instance loses its network connection to AWS, tasks that are running on your instances continue to run anyway unless stopped manually. After the connection to AWS is restored, the AWS credentials that are used by the Amazon ECS container agent and Systems Manager Agent on the external instance renew automatically. For more information about the AWS domains that are used for communication between your external instance and AWS, see [Networking with ECS Anywhere \(p. 416\)](#).

Issues running tasks on your external instance

If your tasks or containers fail to run on your external instance, the most common causes are either network or permission related. If your containers are pulling their images from Amazon ECR or are configured to send container logs to CloudWatch Logs, your task definition must specify a valid task execution IAM role. Without a valid task execution IAM role, your containers will fail to start. For more information, see [Conditional IAM permissions \(p. 418\)](#). For more information about network related issues, see [External instance network issues \(p. 815\)](#).

Important

Amazon ECS provides the Amazon ECS logs collection tool. You can use it to collect logs from your external instances for troubleshooting purposes. For more information, see [Amazon ECS logs collector \(p. 832\)](#).

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 \(p. 636\)](#).

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.

To check stopped tasks for errors (AWS Management 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

Cluster	default
Container Instance	dd3599e9-2ca6-40f4-9da5-a0bb10408260
EC2 instance id	i-83c6ab47
Task Definition	curler:4
Last status	STOPPED
Desired status	STOPPED
Created at	2015-11-20 13:31:01 -0800
Stopped at	2015-11-20 13:31:03 -0800
Stopped reason	Essential container in task exited

Some possible reasons and their explanations are listed below:

Task failed ELB health checks in (elb elb-name)

The current task failed the Elastic Load Balancing health check for the load balancer that's associated with the task's service. For more information, see [Troubleshooting service load balancers \(p. 826\)](#).

Scaling activity initiated by (deployment deployment-id)

When you reduce the desired count of a stable service, some tasks must be stopped to reach the desired number. Tasks that are stopped by downscaling services have this stopped reason.

Host EC2 (instance `id`) stopped/terminated

If you stop or terminate a container instance with running tasks, then the tasks are given this stopped reason.

Container instance deregistration forced by user

If you force the deregistration of a container instance with running tasks, then the tasks are given this stopped reason.

Essential container in task exited

If a container marked as `essential` in task definitions exits or dies, that can cause a task to stop. When an essential container exiting is the cause of a stopped task, the [Step 6 \(p. 816\)](#) can provide more diagnostic information about why the container stopped.

6. If you have a container that has stopped, expand the container and inspect the **Status reason** row to see what caused the task state to change.

Containers

	Name	Container Id	Status
▼	curler	3f871451-c9f1-4d6f-a...	STOPPED (CannotPullContainerError: Error: image tutum/bogus)

Details

```
Status reason CannotPullContainerError: Error: image tutum/bogus:latest not found
Command["/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, you can connect to the container instance with SSH and inspect the Docker container locally. For more information, see [Inspect Docker Containers \(p. 836\)](#).

To check stopped tasks for errors (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
```

CannotPullContainer task errors

The following errors indicate that, when creating a task, the container image specified can't be retrieved.

Connection timed out

When a Fargate task is launched, its elastic network interface requires a route to the internet to pull container images. If you receive an error similar to the following when launching a task, it'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 \(p. 510\)](#).

- For tasks in private subnets, specify **DISABLED** for **Auto-assign public IP** when launching the task, and configure a NAT gateway in your VPC to route requests to the internet. For more information, see [NAT Gateways](#) in the *Amazon VPC User Guide*. For instructions on how to create a VPC with public and private subnets, including a NAT gateway for the private subnets, see [Tutorial: Creating a VPC with Public and Private Subnets for Your Clusters \(p. 727\)](#).

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 \(p. 691\)](#).

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/GetImageBlob11111111: 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
STATUS	PORTS	NAMES	
50501b5f4cbf	amazon/amazon-ecs-agent:latest	"/agent"	4 days ago
Up 4 days		ecs-agent	

By default, when using the `json-file` log driver, Docker captures the standard output (and standard error) of all of your containers and writes them in files using the JSON format. You can set the `max-size` as a log driver option, which prevents the log file from taking up too much space. For more information, see [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-opt": {  
        "max-size": "256m"  
    }  
}
```

An alternative if your container logs are taking up too much disk space is to use the `awslogs` log driver. The `awslogs` log driver sends the logs to CloudWatch, which frees up the disk space that would otherwise be used for your container logs on the container instance. For more information, see [Using the awslogs log driver \(p. 283\)](#).

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](#).

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.

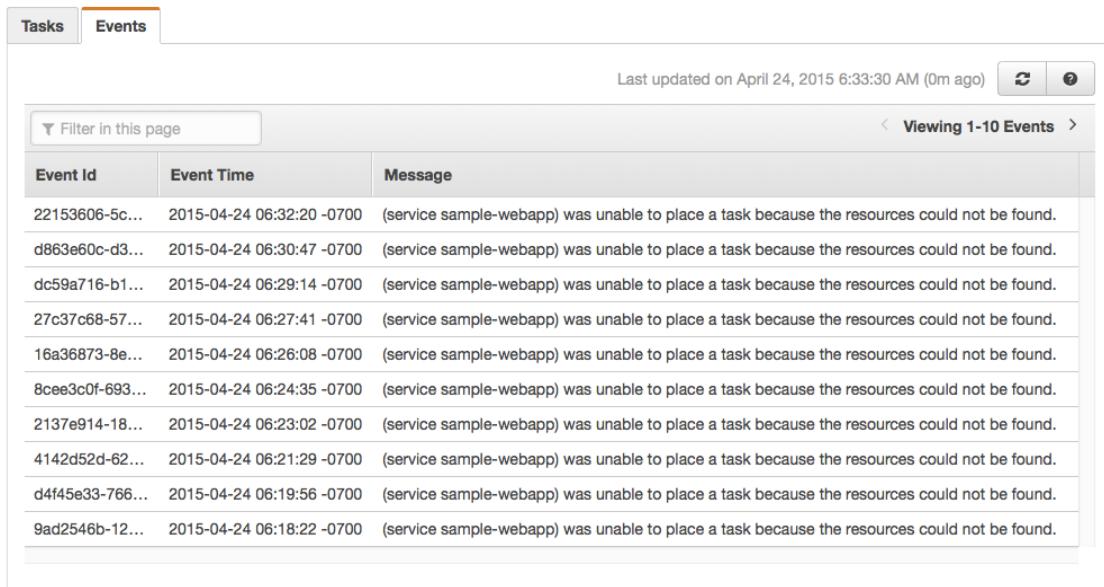
- Task placement and instance health events from the Amazon ECS service scheduler. These events will have a prefix of **service (*service-name*)**. To ensure that this event view is helpful, we only show the 100 most recent events and duplicate event messages are omitted until either the cause is resolved 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.

Viewing service event messages (AWS Management Console)

To view the service event log in the Amazon ECS console

1. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
2. On the **Clusters** page, select the cluster in which your service resides.
3. On the **Cluster : *clustername*** page, select the service to inspect.
4. On the **Service : *servicename*** page, choose **Events**.



Event Id	Event Time	Message
22153606-5c...	2015-04-24 06:32:20 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
d863e60c-d3...	2015-04-24 06:30:47 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
dc59a716-b1...	2015-04-24 06:29:14 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
27c37c68-57...	2015-04-24 06:27:41 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
16a36873-8e...	2015-04-24 06:26:08 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
8cee3c0f-693...	2015-04-24 06:24:35 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
2137e914-18...	2015-04-24 06:23:02 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
4142d52d-62...	2015-04-24 06:21:29 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
d4f45e33-766...	2015-04-24 06:19:56 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.
9ad2546b-12...	2015-04-24 06:18:22 -0700	(service sample-webapp) was unable to place a task because the resources could not be found.

5. Examine the **Message** column for errors or other helpful information.

Viewing service event messages (AWS CLI)

Use the [describe-services](#) command to view the service event messages for a specified service.

The following AWS CLI example describes the *service-name* service in the *default* cluster, which will provide the latest service event messages.

```
aws ecs describe-services \
--cluster default \
--services service-name \
--region us-west-2
```

Service event messages

The following are examples of service event messages you may see in the Amazon ECS console.

service (*service-name*) has reached a steady state.

The service scheduler will send a service (*service-name*) has reached a steady state. service event when the service is healthy and at the desired number of tasks, thus reaching a steady state.

service (*service-name*) was unable to place a task because no container instance met all of its requirements.

The service scheduler will send this event message when it could not find the available resources to add another task. The possible causes for this are:

No container instances were found in your cluster

If no container instances are registered in the cluster you attempt to run a task in, you will receive this error. You should add container instances to your cluster. For more information, see [Launching an Amazon ECS Linux container instance \(p. 364\)](#).

Not enough ports

If your task uses fixed host port mapping (for example, your task uses port 80 on the host for a web server), you must have at least one container instance per task, because only one container can use a single host port at a time. You should add container instances to your cluster or reduce your number of desired tasks.

Too many ports registered

The closest matching container instance for task placement can not exceed the maximum allowed reserved port limit of 100 host ports per container instance. Using dynamic host port mapping may remediate the issue.

Port already in-use

The task definition of this task uses the same port in its port mapping as an task already running on the container instance that was chosen to run on. The service event message would have the chosen container instance ID as part of the message below.

The closest matching container-instance is already using a port required by your task.

Not enough memory

If your task definition specifies 1000 MiB of memory, and the container instances in your cluster each have 1024 MiB of memory, you can only run one copy of this task per container instance. You can experiment with less memory in your task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

Note

If you are trying to maximize your resource utilization by providing your tasks as much memory as possible for a particular instance type, see [Container Instance Memory Management \(p. 382\)](#).

Not enough CPU

A container instance has 1,024 CPU units for every CPU core. If your task definition specifies 1,000 CPU units, and the container instances in your cluster each have 1,024 CPU units, you can only run one copy of this task per container instance. You can experiment with fewer CPU units in your

task definition so that you could launch more than one task per container instance, or launch more container instances into your cluster.

Not enough available ENI attachment points

Tasks that use the `awsvpc` network mode each receive their own elastic network interface (ENI), which is attached to the container instance that hosts it. Amazon EC2 instances have a limit to the number of ENIs that can be attached to them and there are no container instances in the cluster that have ENI capacity available.

The ENI limit for individual container instances depends on the following conditions:

- If you **have not** opted in to the `awsvpcTrunking` account setting, the ENI limit for each container instance depends on the instance type. For more information, see [IP Addresses Per Network Interface Per Instance Type in the Amazon EC2 User Guide for Linux Instances](#).
- If you **have** opted in to the `awsvpcTrunking` account setting but you **have not** launched new container instances using a supported instance type after opting in, the ENI limit for each container instance will still be at the default value. For more information, see [IP Addresses Per Network Interface Per Instance Type in the Amazon EC2 User Guide for Linux Instances](#).
- If you **have** opted in to the `awsvpcTrunking` account setting and you **have** launched new container instances using a supported instance type after opting in, additional ENIs are available. For more information, see [Supported Amazon EC2 instance types \(p. 375\)](#).

For more information about opting in to the `awsvpcTrunking` account setting, see [Elastic network interface trunking \(p. 372\)](#).

You can add container instances to your cluster to provide more available network adapters.

Container instance missing required attribute

Some task definition parameters require a specific Docker remote API version to be installed on the container instance. Others, such as the logging driver options, require the container instances to register those log drivers with the `ECS_AVAILABLE_LOGGING_DRIVERS` agent configuration variable. If your task definition contains a parameter that requires a specific container instance attribute, and you do not have any available container instances that can satisfy this requirement, the task cannot be placed.

A common cause of this error is if your service is using tasks that use the `awsvpc` network mode and the EC2 launch type and the cluster you specified does not have a container instance registered to it in the same subnet that was specified in the `awsvpcConfiguration` when the service was created.

For more information on which attributes are required for specific task definition parameters and agent configuration variables, see [Task definition parameters \(p. 209\)](#) and [Amazon ECS container agent configuration \(p. 454\)](#).

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* has insufficient CPU units available.

The closest matching container instance for task placement does not contain enough CPU units to meet the requirements in the task definition. Review the CPU requirements in both the task size and container definition parameters of the task definition.

service (*service-name*) was unable to place a task because no container instance met all of its requirements. The closest matching container-instance *container-instance-id* encountered error "AGENT".

The Amazon ECS container agent on the closest matching container instance for task placement is disconnected. If you can connect to the container instance with SSH, you can examine the agent logs;

for more information, see [Amazon ECS Container Agent Log \(p. 829\)](#). You should also verify that the agent is running on the instance. If you are using the Amazon ECS-optimized AMI, you can try stopping and restarting the agent with the following command.

- For the Amazon ECS-optimized Amazon Linux 2 AMI

```
sudo systemctl restart ecs
```

- For the Amazon ECS-optimized Amazon Linux AMI

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

service (*service-name*) (instance *instance-id*) is unhealthy in (elb *elb-name*) due to (reason Instance has failed at least the UnhealthyThreshold number of health checks consecutively.)

This service is registered with a load balancer and the load balancer health checks are failing. For more information, see [Troubleshooting service load balancers \(p. 826\)](#).

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 \(p. 602\)](#).

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 is four then 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.

The following are the list of error messages:

- This service is unable to place the task because you have reached the limit on the number of tasks.
- This service is unable to place a task. Reason: The requested CPU configuration is above your limit.
- This service is 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" \(p. 611\)](#). 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" \(p. 614\)](#).

For information about how to view the subnet Availability Zone, see [View your subnet](#) in the *Amazon VPC 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)
512 (.5 vCPU)	1024 (1GB), 2048 (2GB), 3072 (3GB), 4096 (4GB)
1024 (1 vCPU)	2048 (2GB), 3072 (3GB), 4096 (4GB), 5120 (5GB), 6144 (6GB), 7168 (7GB), 8192 (8GB)
2048 (2 vCPU)	Between 4096 (4GB) and 16384 (16GB) in increments of 1024 (1GB)
4096 (4 vCPU)	Between 8192 (8GB) and 30720 (30GB) in increments of 1024 (1GB)

CannotCreateContainerError: API error (500): devmapper

The following Docker error indicates that the thin pool storage on your container instance is full, and that the Docker daemon cannot create new containers:

```
CannotCreateContainerError: API error (500): devmapper: Thin Pool has 4350 free data blocks which is less than minimum required 4454 free data blocks. Create more free space in thin pool or use dm.min_free_space option to change behavior
```

By default, Amazon ECS-optimized Amazon Linux AMIs from version 2015.09.d and later launch with an 8-GiB volume for the operating system that's attached at `/dev/xvda` and mounted as the root of the file system. There's an additional 22-GiB volume that's attached at `/dev/xvdcz` that Docker uses for image and metadata storage. If this storage space is filled up, the Docker daemon cannot create new containers.

The easiest way to add storage to your container instances is to terminate the existing instances and launch new ones with larger data storage volumes. However, if you can't do this, you can add storage to the volume group that Docker uses and extend its logical volume by following the procedures in [AMI storage configuration \(p. 357\)](#).

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

- To view the thin poll information, run the following command on your container instance:

```
docker info
```

- (Amazon ECS container agent 1.8.0 and later) Reduce the amount of time that stopped or exited containers remain on your container instances. The `ECS_ENGINE_TASK_CLEANUP_WAIT_DURATION`

agent configuration variable sets the time duration to wait from when a task is stopped until the Docker container is removed (by default, this value is 3 hours). This removes the Docker container data. If this value is set too low, you may not be able to inspect your stopped containers or view the logs before they are removed. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

- Remove non-running containers and unused images from your container instances. You can use the following example commands to manually remove stopped containers and unused images. Deleted containers can't be inspected later, and deleted images must be pulled again before starting new containers from them.

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

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

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

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

- Remove unused data blocks within containers. You can use the following command to run `fstrim` on any running container and discard any data blocks that are unused by the container file system.

```
sudo sh -c "docker ps -q | xargs docker inspect --format='{{ .State.Pid }}' | xargs -IZ fstrim /proc/z/root/"
```

Troubleshooting service load balancers

Amazon ECS services can register tasks with an Elastic Load Balancing load balancer. Load balancer configuration errors are common causes for stopped tasks. If your stopped tasks were started by services that use a load balancer, consider the following possible causes.

Important

Container health checks 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.

Amazon ECS service-linked role doesn't exist

The Amazon ECS service-linked role allows Amazon ECS services to register container instances with Elastic Load Balancing load balancers. The service-linked role must be created in your account. For more information, see [Service-linked role for Amazon ECS \(p. 685\)](#).

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.

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](#).

Enabling Docker debug output

If you are having trouble with Docker containers or images, you can enable debug mode on your Docker daemon. Enabling debugging provides more verbose output from the daemon and you can use this information to find out more about why your containers or images are having issues.

Enabling Docker debug mode can be especially useful in retrieving error messages that are sent from container registries, such as Amazon ECR, and, in many circumstances, enabling debug mode is the only way to see these error messages.

Important

This procedure is written for the Amazon ECS-optimized Amazon Linux AMI. For other operating systems, see [Enable debugging](#) and [Control and configure Docker with systemd](#) in the Docker documentation.

To enable Docker daemon debug mode on the Amazon ECS-optimized Amazon Linux AMI

1. Connect to your container instance. For more information, see [Connect to your container instance \(p. 384\)](#).
2. Open the Docker options file with a text editor, such as **vi**. For the Amazon ECS-optimized Amazon Linux AMI, the Docker options file is at `/etc/sysconfig/docker`.
3. Find the Docker options statement and add the `-D` option to the string, inside the quotes.

Note

If the Docker options statement begins with a `#`, remove that character to uncomment the statement and enable the options.

For the Amazon ECS-optimized Amazon Linux AMI, the Docker options statement is called `OPTIONS`. For example:

```
# Additional startup options for the Docker daemon, for example:  
# OPTIONS="--ip-forward=true --iptables=true"  
# By default we limit the number of open files per container  
OPTIONS="-D --default-ulimit nofile=1024:4096"
```

4. Save the file and exit your text editor.
5. Restart the Docker daemon.

```
sudo service docker restart
```

The output is as follows:

```
Stopping docker: [ OK ]  
Starting docker: . [ OK ]
```

6. Restart the Amazon ECS agent.

```
sudo service ecs restart
```

Your Docker logs should now show more verbose output.

```
time="2015-12-30T21:48:21.907640838Z" level=debug msg="Unexpected response from server: \"{\\"errors\\\":[{\\"code\\\":\"DENIED\\\", \\"message\\\":\"User: arn:aws:sts::1111:assumed-role/ecrReadOnly/i-abcdefg is not authorized to perform: ecr:InitiateLayerUpload on resource: arn:aws:ecr:us-east-1:1111:repository/nginx_test \\"}]}\\n\" http.Header{\"Connection\":[]string{\"keep-alive\"}, \"Content-Type\": []string{\"application/json; charset=utf-8\"}, \"Date\":[]string{\"Wed, 30 Dec 2015 21:48:21 GMT\"}, \"Docker-Distribution-Api-Version\":[]string{\"registry/2.0\"}, \"Content-Length\":[]string{\"235\"}}"
```

Amazon ECS Log File Locations

Amazon ECS stores logs in the `/var/log/ecs` folder of your container instances. There are logs available from the Amazon ECS container agent and from the `ecs-init` service that controls the state

of the agent (start/stop) on the container instance. You can view these log files by connecting to a container instance using SSH. For more information, see [Connect to your container instance \(p. 384\)](#).

Note

If you are not sure how to collect all of the logs on your container instances, you can use the Amazon ECS logs collector. For more information, see [Amazon ECS logs collector \(p. 832\)](#).

Amazon ECS Container Agent Log

The Amazon ECS container agent stores logs on your container instances.

For container agent version 1.36.0 and later, by default the logs are located at `/var/log/ecs/ecs-agent.log` on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log` on Windows instances.

For container agent version 1.35.0 and earlier, by default the logs are located at `/var/log/ecs/ecs-agent.log.timestamp` on Linux instances and at `C:\ProgramData\Amazon\ECS\log\ecs-agent.log.timestamp` on Windows instances.

By default, the agent logs are rotated hourly with a maximum of 24 logs being stored.

The following are the container agent configuration variables that can be used to change the default agent logging behavior. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

`ECS_LOGFILE`

Example values: `/ecs-agent.log`

Default value on Linux: Null

Default value on Windows: Null

The location where agent logs should be written. If you are running the agent via `ecs-init`, which is the default method when using the Amazon ECS-optimized AMI, the in-container path will be `/log` and `ecs-init` mounts that out to `/var/log/ecs/` on the host.

`ECS_LOGLEVEL`

Example values: `crit, error, warn, info, debug`

Default value on Linux: `info`

Default value on Windows: `info`

The level of detail to log.

`ECS_LOGLEVEL_ON_INSTANCE`

Example values: `none, crit, error, warn, info, debug`

Default value on Linux: `none`, if `ECS_LOG_DRIVER` is explicitly set to a non-empty value; otherwise the same value as `ECS_LOGLEVEL`

Default value on Windows: `none`, if `ECS_LOG_DRIVER` is explicitly set to a non-empty value; otherwise the same value as `ECS_LOGLEVEL`

Can be used to override `ECS_LOGLEVEL` and set a level of detail that should be logged in the on-instance log file, separate from the level that is logged in the logging driver. If a logging driver is explicitly set, on-instance logs are turned off by default, but can be turned back on with this variable.

ECS_LOG_ROLLOVER_TYPE

Example values: `size`, `hourly`

Default value on Linux: `hourly`

Default value on Windows: `hourly`

Determines whether the container agent log file will be rotated hourly or based on size. By default, the agent log file is rotated each hour.

ECS_LOG_OUTPUT_FORMAT

Example values: `logfmt`, `json`

Default value on Linux: `logfmt`

Default value on Windows: `logfmt`

Determines the log output format. When the `json` format is used, each line in the log will be a structured JSON map.

ECS_LOG_MAX_FILE_SIZE_MB

Example values: 10

Default value on Linux: 10

Default value on Windows: 10

When the `ECS_LOG_ROLLOVER_TYPE` variable is set to `size`, this variable determines the maximum size (in MB) of the log file before it is rotated. If the rollover type is set to `hourly`, then this variable is ignored.

ECS_LOG_MAX_ROLL_COUNT

Example values: 24

Default value on Linux: 24

Default value on Windows: 24

Determines the number of rotated log files to keep. Older log files are deleted after this limit is reached.

For container agent version 1.36.0 and later, the following is an example log file when the `logfmt` format is used.

```
level=info time=2019-12-12T23:43:29Z msg="Loading configuration" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-ecs-agent:latest" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Image excluded from cleanup: amazon/amazon-ecs-pause:0.1.0" module=parse.go
level=info time=2019-12-12T23:43:29Z msg="Amazon ECS agent Version: 1.36.0, Commit: ca640387" module=agent.go
level=info time=2019-12-12T23:43:29Z msg="Creating root ecs cgroup: /ecs" module=init_linux.go
level=info time=2019-12-12T23:43:29Z msg="Creating cgroup /ecs" module=cgroup_controller_linux.go
level=info time=2019-12-12T23:43:29Z msg="Loading state!" module=statemanager.go
```

```
level=info time=2019-12-12T23:43:29Z msg="Event stream ContainerChange start listening..."  
module=eventstream.go  
level=info time=2019-12-12T23:43:29Z msg="Restored cluster 'auto-robc'" module=agent.go  
level=info time=2019-12-12T23:43:29Z msg="Restored from checkpoint file. I  
am running as 'arn:aws:ecs:us-west-2:0123456789:container-instance/auto-  
robc/3330a8a91d15464ea30662d5840164cd' in cluster 'auto-robc'" module=agent.go
```

The following is an example log file when the JSON format is used.

```
{"time": "2019-11-07T22:52:02Z", "level": "info", "msg": "Starting Amazon Elastic Container  
Service Agent", "module": "engine.go"}
```

For container agent versions 1.35.0 and earlier, the following is the format of the log file.

```
2016-08-15T15:54:41Z [INFO] Starting Agent: Amazon ECS Agent - v1.12.0 (895f3c1)  
2016-08-15T15:54:41Z [INFO] Loading configuration  
2016-08-15T15:54:41Z [WARN] Invalid value for task cleanup duration, will be overridden to  
3h0m0s, parsed value 0, minimum threshold 1m0s  
2016-08-15T15:54:41Z [INFO] Checkpointing is enabled. Attempting to load state  
2016-08-15T15:54:41Z [INFO] Loading state! module="statemanager"  
2016-08-15T15:54:41Z [INFO] Detected Docker versions [1.17 1.18 1.19 1.20 1.21 1.22]  
2016-08-15T15:54:41Z [INFO] Registering Instance with ECS  
2016-08-15T15:54:41Z [INFO] Registered! module="api client"
```

Amazon ECS ecs-init Log

The `ecs-init` process stores logs at `/var/log/ecs/ecs-init.log`.

```
cat /var/log/ecs/ecs-init.log
```

Output:

```
2018-02-16T18:13:54Z [INFO] pre-start  
2018-02-16T18:13:56Z [INFO] start  
2018-02-16T18:13:56Z [INFO] No existing agent container to remove.  
2018-02-16T18:13:56Z [INFO] Starting Amazon Elastic Container Service Agent
```

IAM Roles for Tasks Credential Audit Log

When the credential provider for the IAM role is used to provide credentials to tasks, these requests are saved in an audit log. The audit log inherits the same log rotation settings as the container agent log. The `ECS_LOG_ROLLOVER_TYPE`, `ECS_LOG_MAX_FILE_SIZE_MB`, and `ECS_LOG_MAX_ROLL_COUNT` container agent configuration variables can be set to affect the behavior of the audit log. For more information, see [Amazon ECS Container Agent Log \(p. 829\)](#).

For container agent version 1.36.0 and later, the audit log is located at `/var/log/ecs/audit.log`. When the log is rotated, a timestamp in `YYYY-MM-DD-HH` format is added to the end of the log file name.

For container agent version 1.35.0 and earlier, the audit log is located at `/var/log/ecs/audit.log`. `YYYY-MM-DD-HH`.

The log entry format is as follows:

- Timestamp

- HTTP response code
- IP address and port number of request origin
- Relative URI of the credential provider
- The user agent that made the request
- The ARN of the task to which the requesting container belongs
- The GetCredentials API name and version number
- The name of the Amazon ECS cluster to which the container instance is registered
- The container instance ARN

An example log entry is shown below.

```
cat /var/log/ecs/audit.log.2016-07-13-16
```

Output:

```
2016-07-13T16:11:53Z 200 172.17.0.5:52444 "/v1/credentials" "python-requests/2.7.0
CPython/2.7.6 Linux/4.4.14-24.50.amzn1.x86_64" TASK_ARN GetCredentials
1 CLUSTER_NAME CONTAINER_INSTANCE_ARN
```

Amazon ECS logs collector

If you are unsure how to collect all of the various logs on your container instances, you can use the Amazon ECS logs collector. It is available on GitHub for both [Linux](#) and [Windows](#). The script collects general operating system logs as well as Docker and Amazon ECS container agent logs, which can be helpful for troubleshooting AWS Support cases. It then compresses and archives the collected information into a single file that can easily be shared for diagnostic purposes. It also supports enabling debug mode for the Docker daemon and the Amazon ECS container agent on Amazon Linux variants, such as the Amazon ECS-optimized AMI. Currently, the Amazon ECS logs collector supports the following operating systems:

- Amazon Linux
- Red Hat Enterprise Linux 7
- Debian 8
- Ubuntu 14.04
- Windows Server 2016

Note

The source code for the Amazon ECS logs collector is available on GitHub for both [Linux](#) and [Windows](#). We encourage you to submit pull requests for changes that you would like to have included. However, Amazon Web Services doesn't currently support running modified copies of this software.

To download and run the Amazon ECS logs collector for Linux

1. Connect to your container instance. For more information, see [Connect to your container instance \(p. 384\)](#).
2. Download the Amazon ECS logs collector script.

```
curl -O https://raw.githubusercontent.com/awslabs/ecs-logs-collector/master/ecs-logs-collector.sh
```

3. Run the script to collect the logs and create the archive.

Note

To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `--mode=enable-debug` option to the command below. This may restart the Docker daemon, which kills all containers that are running on the instance. Consider draining the container instance and moving any important tasks to other container instances before enabling debug mode. For more information, see [Container instance draining \(p. 428\)](#).

```
[ec2-user ~]$ sudo bash ./ecs-logs-collector.sh
```

After you have run the script, you can examine the collected logs in the `collect` folder that the script created. The `collect.tgz` file is a compressed archive of all of the logs, which you can share with AWS Support for diagnostic help.

To download and run the Amazon ECS logs collector for Windows

1. Connect to your container instance. For more information, see [Connecting to Your Windows Instance](#) in the [Amazon EC2 User Guide for Windows Instances](#).
2. Download the Amazon ECS logs collector script using PowerShell.

```
Invoke-WebRequest -OutFile ecs-logs-collector.ps1 https://raw.githubusercontent.com/awslabs/aws-ecs-logs-collector-for-windows/master/ecs-logs-collector.ps1
```

3. Run the script to collect the logs and create the archive.

Note

To enable the debug mode for the Docker daemon and the Amazon ECS container agent, add the `-RunMode debug` option to the command below. This restarts the Docker daemon, which kills all containers that are running on the instance. Consider draining the container instance and moving any important tasks to other container instances before enabling debug mode. For more information, see [Container instance draining \(p. 428\)](#).

```
.\ecs-logs-collector.ps1
```

After you have run the script, you can examine the collected logs in the `collect` folder that the script created. The `collect.tgz` file is a compressed archive of all of the logs, which you can share with AWS Support for diagnostic help.

Agent introspection diagnostics

The Amazon ECS agent introspection API can provide helpful diagnostic information. For example, you can use the agent introspection API to get the Docker ID for a container in your task. You can use the agent introspection API by connecting to a container instance using SSH. For more information, see [Connect to your container instance \(p. 384\)](#).

Important

Your container instance must have an IAM role that allows access to Amazon ECS in order to reach the introspection API. For more information, see [Amazon ECS container instance IAM role \(p. 695\)](#).

The below example shows two tasks, one that is currently running and one that was stopped.

Note

The command below is piped through the **python -mjson.tool** for greater readability.

```
curl http://localhost:51678/v1/tasks | python -mjson.tool
```

Output:

```
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
100  1095  100  1095      0       0  117k      0  ---:---:---:---:---:---:---:--- 133k
{
  "Tasks": [
    {
      "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/090eff9b-1ce3-4db6-848a-a8d14064fd24",
      "Containers": [
        {
          "DockerId": "189a8ff4b5f04affe40e5160a5ffadca395136eb5faf4950c57963c06f82c76d",
          "DockerName": "ecs-console-sample-app-static-6-simple-app-86caf9bcabe3e9c61600",
          "Name": "simple-app"
        },
        {
          "DockerId": "f7f1f8a7a245c5da83aa92729bd28c6bcb004d1f6a35409e4207e1d34030e966",
          "DockerName": "ecs-console-sample-app-static-6-busybox-ce83ce978a87a890ab01",
          "Name": "busybox"
        }
      ],
      "Family": "console-sample-app-static",
      "KnownStatus": "STOPPED",
      "Version": "6"
    },
    {
      "Arn": "arn:aws:ecs:us-west-2:aws_account_id:task/1810e302-eaea-4da9-a638-097bea534740",
      "Containers": [
        {
          "DockerId": "dc7240fe892ab233dbbcee5044d95e1456c120dba9a6b56ec513da45c38e3aeb",
          "DockerName": "ecs-console-sample-app-static-6-simple-app-f0e5859699a7aecfb101",
          "Name": "simple-app"
        },
        {
          "DockerId": "096d685fb85a1ff3e021c8254672ab8497e3c13986b9cf005cbae9460b7b901e",
          "DockerName": "ecs-console-sample-app-static-6-busybox-92e4b8d0ecd0cce69a01",
          "Name": "busybox"
        }
      ],
      "DesiredStatus": "RUNNING",
      "Family": "console-sample-app-static",
      "KnownStatus": "RUNNING",
      "Version": "6"
    }
  ]
}
```

In the above example, the stopped task (`090eff9b-1ce3-4db6-848a-a8d14064fd24`) has two containers. You can use `docker inspect container-ID` to view detailed information on each container. For more information, see [Amazon ECS container agent introspection \(p. 503\)](#).

Docker diagnostics

Docker provides several diagnostic tools that help you troubleshoot problems with your containers and tasks. For more information about all of the available Docker command line utilities, see the [Docker Command Line](#) topic in the Docker documentation. You can access the Docker command line utilities by connecting to a container instance using SSH. For more information, see [Connect to your container instance \(p. 384\)](#).

The exit codes that Docker containers report can also provide some diagnostic information (for example, exit code 137 means that the container received a `SIGKILL` signal). For more information, see [Exit Status](#) in the Docker documentation.

List Docker containers

You can use the `docker ps` command on your container instance to list the running containers. In the below example, only the Amazon ECS container agent is running. For more information, see [docker ps](#) in the Docker documentation.

```
docker ps
```

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
cee0d6986de0	amazon/amazon-ecs-agent:latest	"/agent"	22 hours ago
Up 22 hours	127.0.0.1:51678->51678/tcp	ecs-agent	

You can use the `docker ps -a` command to see all containers (even stopped or killed containers). This is helpful for listing containers that are unexpectedly stopping. In the following example, container `f7f1f8a7a245` exited 9 seconds ago, so it doesn't show up in a `docker ps` output without the `-a` flag.

```
docker ps -a
```

Output:

CONTAINER ID	IMAGE	COMMAND	
CREATED	STATUS	PORTS	NAMES
db4d48e411b1	amazon/ecs-emptyvolume-base:autogenerated	"not-applicable"	19
seconds ago			ecs-console-
f7f1f8a7a245	busybox:buildroot-2014.02	"\sh -c '/bin/sh -c	22
hours ago	Exited (137) 9 seconds ago		ecs-console-
sample-app-static-6-internalecs-emptyvolume-source-c09288a6b0cba8a53700			
189a8ff4b5f0	httpd:2	"httpd-foreground"	22
hours ago	Exited (137) 40 seconds ago		ecs-console-
sample-app-static-6-simple-app-86caf9bcabe3e9c61600			
0c7dca9321e3	amazon/ecs-emptyvolume-base:autogenerated	"not-applicable"	22
hours ago			ecs-console-
sample-app-static-6-internalecs-emptyvolume-source-90fefaa68498a8a80700			
cee0d6986de0	amazon/amazon-ecs-agent:latest	"/agent"	22
hours ago	Up 22 hours	127.0.0.1:51678->51678/tcp	ecs-agent

View Docker Logs

You can view the `STDOUT` and `STDERR` streams for a container with the `docker logs` command. In this example, the logs are displayed for the `dc7240fe892a` container and piped through the `head` command for brevity. For more information, go to [docker logs](#) in the Docker documentation.

Note

Docker logs are only available on the container instance if you are using the default `json` log driver. If you have configured your tasks to use the `awslogs` log driver, then your container logs are available in CloudWatch Logs. For more information, see [Using the awslogs log driver \(p. 283\)](#).

```
docker logs dc7240fe892a | head
```

Output:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.11. Set the 'ServerName' directive globally to suppress this message
[Thu Apr 23 19:48:36.956682 2015] [mpm_event:notice] [pid 1:tid 140327115417472] AH00489:
Apache/2.4.12 (Unix) configured -- resuming normal operations
[Thu Apr 23 19:48:36.956827 2015] [core:notice] [pid 1:tid 140327115417472] AH00094:
Command line: 'httpd -D FOREGROUND'
10.0.1.86 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:48:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:29 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.0.154 - - [23/Apr/2015:19:49:50 +0000] "-" 408 -
10.0.1.86 - - [23/Apr/2015:19:49:58 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:49:59 +0000] "GET / HTTP/1.1" 200 348
10.0.1.86 - - [23/Apr/2015:19:50:28 +0000] "GET / HTTP/1.1" 200 348
10.0.0.154 - - [23/Apr/2015:19:50:29 +0000] "GET / HTTP/1.1" 200 348
time="2015-04-23T20:11:20Z" level="fatal" msg="write /dev/stdout: broken pipe"
```

Inspect Docker Containers

If you have the Docker ID of a container, you can inspect it with the `docker inspect` command. Inspecting containers provides the most detailed view of the environment in which a container was launched. For more information, see [docker inspect](#) in the Docker documentation.

```
docker inspect dc7240fe892a
```

Output:

```
[{
    "AppArmorProfile": "",
    "Args": [],
    "Config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "httpd-foreground"
        ],
        "CpuShares": 10,
        "Cpuset": ""
    },
    "HostConfig": {
        "Binds": []
    }
}]
```

```
"Domainname": "",  
"Entrypoint": null,  
"Env": [  
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/  
apache2/bin",  
    "HTTPD_PREFIX=/usr/local/apache2",  
    "HTTPD_VERSION=2.4.12",  
    "HTTPD_BZ2_URL=https://www.apache.org/dist/httpd/httpd-2.4.12.tar.bz2"  
,  
"ExposedPorts": {  
    "80/tcp": {}  
},  
"Hostname": "dc7240fe892a",  
...  
]
```

AWS Fargate throttling limits

AWS Fargate throttles Amazon ECS RunTask API requests for each AWS account on a per-Region basis. We do this to help the performance of the service, and to ensure fair usage for all Fargate customers. Throttling ensures that calls to the launch tasks do not exceed the maximum allowed API request limits. API calls are subject to the request limits whether they originate from the Amazon ECS console, a command line tool, or a third-party application.

When the RunTask API throttling limit is reached, you will get the following exception:

```
An error occurred (ThrottlingException) when calling the RunTask operation (reached max  
retries: 4): Rate exceeded.
```

To request an API rate limit quota increase, complete the following steps.

To request a rate limit increase

1. Open the [AWS Support Center](#).
2. Choose **Create case, Service limit increase**.
3. For **Limit type**, select **Fargate**.
4. For **Region**, select the Region that you want to submit the rate limit increase for.
5. For **Limit**, choose **Concurrent Task Limit**.
6. For **Use case description**, describe your use case concerning the RunTask API throttle limit so support can assist you further.

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*.

API action	Failure reason	Cause
DescribeClusters	MISSING	The specified cluster wasn't found. Verify the spelling of the cluster name.
DescribeInstances	MISSING	The specified container instance wasn't found. Verify that you specified the cluster the container instance is registered to and that both the container instance ARN or ID is correct.
DescribeServices	MISSING	The specified service wasn't found. Verify that the correct cluster or Region is specified and that the service ARN or name is valid.
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.
RunTask or StartTask	RESOURCE : *	<p>The resource or resources that are requested by the task are unavailable on the container instances in the cluster. If the resource is CPU, memory, ports, or elastic network interfaces, you might need to add additional container instances to your cluster.</p> <p>For RESOURCE : ENI errors, your cluster doesn't have any available elastic network interface attachment points, which are required for tasks that use the awsvpc network mode. Amazon EC2 instances have a limit to the number of network interfaces that can be attached to them, and the primary network interface counts as one. For more information about how many network interfaces are supported for each instance type, see IP Addresses Per Network Interface Per Instance Type in the <i>Amazon EC2 User Guide for Linux Instances</i>.</p> <p>For RESOURCE : GPU errors, the number of GPUs requested by the task are unavailable and you might need to add GPU-enabled container instances to your</p>

API action	Failure reason	Cause
		cluster. For more information, see Working with GPUs on Amazon ECS (p. 250) .
	AGENT	The container instance that you attempted to launch a task onto has an agent that's currently disconnected. To prevent extended wait times for task placement, the request was rejected.
	LOCATION	The container instance that you attempted to launch a task onto is in a different Availability Zone than the subnets that you specified in your <code>awsVpcConfiguration</code> .
	ATTRIBUTE	Your task definition contains a parameter that requires a specific container instance attribute that isn't available on your container instances. For example, if your task uses the <code>awsvpc</code> network mode, but there are no instances in your specified subnets with the <code>ecs.capability.task-eni</code> attribute. For more information about which attributes are required for specific task definition parameters and agent configuration variables, see Task definition parameters (p. 209) and Amazon ECS container agent configuration (p. 454) .
StartTask	MISSING	The container instance you attempted to launch the task onto can't be found. Perhaps the wrong cluster or Region is specified, or the container instance ARN or ID is misspelled.
	INACTIVE	The container instance that you attempted to launch a task onto was previously deregistered with Amazon ECS and can't be used.

Troubleshooting IAM Roles for Tasks

If you are having trouble configuring IAM roles for tasks in your cluster, you can try this known good configuration to help debug your own configuration.

The following procedure helps you to:

- Create a CloudWatch Logs log group to store your test logs.
- Create a task IAM role that has full Amazon ECS permissions.
- Register a task definition with a known good AWS CLI configuration that is compatible with IAM roles for tasks.
- Run a task from that task definition to test your container instance support for IAM roles for tasks.
- View the container logs from that task in CloudWatch Logs to verify that it works.

To test IAM roles for tasks with a known good configuration

1. Create a CloudWatch Logs log group called `ecs-tasks`.
 - a. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
 - b. In the left navigation pane, choose **Logs, Actions, Create log group**.
 - c. For **Log Group Name**, enter `ecs-tasks` and choose **Create log group**.
2. Create an IAM role for your task to use.
 - a. Open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. In the navigation pane, choose **Roles, Create role**.
 - c. For **Select type of trusted entity**, choose **Elastic Container Service**.
 - d. For **Select your use case**, choose **Elastic Container Service Task, Next: Permissions**.
 - e. On the **Attached permissions policy** page, choose **AmazonECS_FullAccess**, **Next: Review**.
 - f. On the **Review** page, for **Role name**, enter `ECS-task-full-access` and choose **Create role**.
3. Register a task definition that uses your new role.
 - a. Open the Amazon ECS console at <https://console.aws.amazon.com/ecs/>.
 - b. In the navigation pane, choose **Task Definitions**.
 - c. On the **Task Definitions** page, choose **Create new Task Definition**.
 - d. On the **Select launch type compatibility** page, choose **EC2, Next step**.
 - e. Scroll to the bottom of the page and choose **Configure via JSON**.
 - f. Paste the sample task definition JSON below into the text area (replacing the pre-populated JSON there) and choose **Save**.

Note

Replace the `awslogs-region` value with the Region where you created your CloudWatch Logs log group.

```
{  
    "taskRoleArn": "ECS-task-full-access",  
    "containerDefinitions": [  
        {  
            "memory": 128,  
            "essential": true,  
            "name": "amazonlinux",  
            "image": "amazonlinux",  
            "entryPoint": [  
                "/bin/bash",  
                "-c"  
            ],  
            "command": [  
                "yum install -y aws-cli; aws ecs list-tasks --region us-west-2"  
            ],  
            "logConfiguration": {  
                "logDriver": "awslogs",  
                "options": {  
                    "awslogs-group": "ecs-tasks",  
                    "awslogs-region": "us-west-2",  
                    "awslogs-stream-prefix": "ecs"  
                }  
            }  
        }  
    ]  
}
```

```

        "options": {
            "awslogs-group": "ecs-tasks",
            "awslogs-region": "us-west-2",
            "awslogs-stream-prefix": "iam-role-test"
        }
    }
],
"family": "iam-role-test",
"requiresCompatibilities": [
    "EC2"
],
"volumes": [],
"placementConstraints": [],
"networkMode": null,
"memory": null,
"cpu": null
}

```

- g. Verify your information and choose **Create**.
4. Run a task from your task definition.
 - a. On the **Task Definition: iam-role-test** registration confirmation page, choose **Actions, Run Task**.
 - b. On the **Run Task** page, choose the **EC2** launch type, a cluster, and then choose **Run Task** to run your task.
5. View the container logs in the CloudWatch Logs console.
 - a. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
 - b. In the left navigation pane, choose **Logs**.
 - c. Select the **ecs-tasks** log group.
 - d. Select the most recent log stream.
 - e. Scroll down to view the last lines of the log stream. You should see the output of the **aws ecs list-tasks** command.

```
{
  "taskArns": [
    "arn:aws:ecs:us-east-1:aws_account_id:task/d48feb62-46e2-4cbc-a36b-e0400b993d1d"
  ]
}
```

If you receive an "Unable to locate credentials" error, then the following are possible causes.

- The IAM roles for tasks feature is not enabled on your container instances. For more information, see [Enabling Task IAM Roles on your Container Instances \(p. 701\)](#).
- The credential URL is being throttled. You can use the **ECS_TASK_METADATA_RPS_LIMIT** container agent parameter to configure the throttle limits. For more information, see [Amazon ECS container agent configuration \(p. 454\)](#).

Amazon EC2 Windows containers

Amazon ECS now supports Windows containers on container instances that are launched with the Amazon ECS-optimized Windows Server AMI.

Windows container instances use their own version of the Amazon ECS container agent. On the Amazon ECS-optimized Windows Server AMI, the Amazon ECS container agent runs as a service on the host. Unlike the Linux platform, the agent doesn't run inside a container because it uses the host's registry and the named pipe at `\.\pipe\docker_engine` to communicate with the Docker daemon.

The source code for the Amazon ECS container agent is [available on GitHub](#). We encourage you to submit pull requests for changes that you would like to have included. However, we do not currently provide support for running modified copies of this software. You can view open issues for Amazon ECS and Windows containers on our [GitHub issues page](#).

Amazon ECS vends AMIs that are optimized for Windows containers in the following variants. For more information, see [Amazon ECS-optimized AMI \(p. 333\)](#).

- **Amazon ECS-optimized Windows Server 2019 Full AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows Server 2019 Core AMI** – Recommended for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows Server 2004 Core AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.
- **Amazon ECS-optimized Windows Server 1909 Core AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.

Important

The Amazon ECS-optimized Windows Server 1909 Core AMI is being deprecated. No new versions of this AMI will be released.

- **Amazon ECS-optimized Windows Server 2016 Full AMI** – Available for launching your Amazon ECS container instances on the Windows operating system.

Windows Server 2022, Windows Server 2019, and Windows Server 2016 are Long-Term Servicing Channel (LTSC) releases. Windows Server 2004 and Windows Server 20H2 are Semi-Annual Channel (SAC) releases. For more information, see [Windows Server release information](#).

Topics

- [Windows container caveats \(p. 842\)](#)
- [Getting started with Windows containers \(p. 843\)](#)
- [Windows task definitions \(p. 843\)](#)
- [Windows IAM roles for tasks \(p. 846\)](#)
- [Pushing Windows images to Amazon ECR \(p. 847\)](#)
- [Building your own Amazon ECS-optimized Windows AMI \(p. 848\)](#)
- [Using gMSAs for Windows Containers \(p. 849\)](#)

Windows container caveats

Here are some things you should know about Amazon EC2 Windows containers and Amazon ECS.

- Windows containers can't run on Linux container instances, and the opposite is also the case. For better task placement for Windows and Linux tasks, keep Windows and Linux container instances in separate clusters and only place Windows tasks on Windows clusters. You can ensure that Windows task definitions are only placed on Windows instances by setting the following placement constraint: `memberOf(ecs.os-type=='windows')`.
- Windows containers are supported for tasks that use the EC2 and Fargate launch types.
- Windows containers and container instances can't support all the task definition parameters that are available for Linux containers and container instances. For some parameters, they aren't supported at all, and others behave differently on Windows than they do on Linux. For more information, see [Windows task definitions \(p. 843\)](#).
- For the IAM roles for tasks feature, you need to configure your Windows container instances to allow the feature at launch. Your containers must run some provided PowerShell code when they use the feature. For more information, see [Windows IAM roles for tasks \(p. 846\)](#).
- The IAM roles for tasks feature uses a credential proxy to provide credentials to the containers. This credential proxy occupies port 80 on the container instance, so if you use IAM roles for tasks, port 80 is not available for tasks. For web service containers, you can use an Application Load Balancer and dynamic port mapping to provide standard HTTP port 80 connections to your containers. For more information, see [Service load balancing \(p. 574\)](#).
- The Windows server Docker images are large (9 GiB). As such, your Windows container instances require more storage space than Linux container instances.

Getting started with Windows containers

Work through a tutorial that guides you through getting Windows containers running on Amazon ECS with the Amazon ECS-optimized Windows Server AMI in the AWS Management Console at [Getting started with the Amazon ECS console using Amazon EC2 Windows containers \(p. 41\)](#).

Windows task definitions

Windows containers and container instances can't support all the task definition parameters that are available for Linux containers and container instances. For some parameters, they aren't supported at all, and others behave differently on Windows than they do on Linux.

Windows task definition parameters

The following list explains which parameters aren't supported or behave differently on Windows containers than they do with Linux containers. For more information about these parameters as they relate to Amazon ECS, see [Task definition parameters \(p. 209\)](#).

`taskRoleArn`

Supported: Yes

IAM roles for tasks on Windows require that the `-EnableTaskIAMRole` option is set when you launch the Amazon ECS-optimized Windows Server AMI. Your containers must also run some configuration code in order to take advantage of the feature. For more information, see [Windows IAM roles for tasks \(p. 846\)](#).

`networkMode`

Supported: Yes

When you register a task definition with Windows containers, you must use `default` or `awsvpc` network mode.

`containerDefinitions`

Supported: Yes

Additional notes: Not all container definition parameters are supported. Review the following list for individual parameter support.

`portMappings`

Supported: Limited

Port mappings on Windows use the `NetNAT` gateway address rather than `localhost`. There is no loopback for port mappings on Windows, so you can't access a container's mapped port from the host itself.

`cpu`

Supported: Yes

Amazon ECS treats this parameter in the same manner that it does for Linux containers: if you provide 500 CPU shares to a container, then 500 CPU shares is removed from the available resources on the container instance when the task is placed. However, on a Windows container instance, the CPU limit is enforced as an absolute limit, or a quota. Windows containers only have access to the specified amount of CPU that is described in the task definition.

`disableNetworking`

Supported: No

`dnsServers`

Supported: No

`dnsSearchDomains`

Supported: No

`dockerSecurityOptions`

Supported: No

`extraHosts`

Supported: No

`links`

Supported: No

`mountPoints`

Supported: Limited

Windows containers can mount whole directories on the same drive as `$env:ProgramData`. Windows containers can't mount directories on a different drive, and mount point can't be across drives.

`linuxParameters`

Supported: No

`privileged`

Supported: No

`readonlyRootFilesystem`

Supported: No

```
user
  Supported: No
ulimits
  Supported: No
volumes
  Supported: Yes
  name
    Supported: Yes
  dockerVolumeConfiguration
    Supported: No
host
  Supported: Limited
  Windows containers can mount whole directories on the same drive as $env:ProgramData.
  Windows containers can't mount directories on a different drive, and mount point can't be
  across drives. For example, you can mount C:\my\path:C:\my\path and D:\:D:\, but not D:
  \my\path:C:\my\path or D:\:C:\my\path.
  FSxWindowsFileServerVolumeConfiguration
    Supported: Yes
    This parameter is specified when you are using the FSx for Windows File Server file system for
    task storage. For more information, see FSx for Windows File Server volumes \(p. 260\).
cpu
  Supported: No
  Task-level CPU is ignored for Windows containers. We recommend specifying container-level CPU for
  Windows containers.
memory
  Supported: No
  Task-level memory is ignored for Windows containers. We recommend specifying container-level
  memory for Windows containers.
proxyConfiguration
  Supported: No
  ipcMode
    Supported: No
  pidMode
    Supported: No
```

Windows sample task definitions

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"  
        }  
    ]  
}
```

Windows IAM roles for tasks

The IAM roles for tasks with Windows features requires extra configuration, but much of this configuration is similar to enabling IAM roles for tasks on Linux container instances. The following requirements must be met to enable IAM roles for tasks for Windows containers.

- When you launch your container instances, you must 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 \(p. 847\)](#).
- You must create an IAM role and policy for your tasks. For more information, see [Creating an IAM Role and Policy for your Tasks \(p. 702\)](#).
- Your container must use an AWS SDK that supports IAM roles for tasks. For more information, see [Using a Supported AWS SDK \(p. 704\)](#).
- 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 \(p. 704\)](#).

- The IAM roles for the task credential provider use port 80 on the container instance. Therefore, if you enable 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 \(p. 574\)](#).
- 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.

```
# Copyright 2014-2016 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You may
# not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

$gateway = (Get-NetRoute | Where { $_.DestinationPrefix -eq '0.0.0.0/0' } | Sort-Object
RouteMetric | Select NextHop).NextHop
$ifIndex = (Get-NetAdapter -InterfaceDescription "Hyper-V Virtual Ethernet*" | Sort-Object
| Select ifIndex)
New-NetRoute -DestinationPrefix 169.254.170.2/32 -InterfaceIndex $ifIndex -NextHop $gateway
# credentials API
New-NetRoute -DestinationPrefix 169.254.169.254/32 -InterfaceIndex $ifIndex -NextHop
$gateway # metadata API
```

Pushing Windows images to Amazon ECR

You can push Windows Docker container images to Amazon ECR. You must be using a version of Docker that supports Windows containers. The following procedures show you how to pull a Windows Docker image, create an Amazon ECR repository to store the image, tag the image to that repository, authenticate the image to the Amazon ECR registry, and then push the image to that repository.

To pull and tag a Windows Docker image

1. Pull a Windows Docker image locally. This example uses the `microsoft/iis` image.

```
PS C:\> docker pull microsoft/iis
Using default tag: latest
latest: Pulling from microsoft/iis

3889bb8d808b: Pull complete
04ee5d718c7a: Pull complete
```

```
c0931dd15237: Pull complete
61784b745c20: Pull complete
d05122f129ca: Pull complete
Digest: sha256:25586570b058da9882d4af640d326d0cc26df60b67e1cee63f35ea54d83c882
Status: Downloaded newer image for microsoft/iis:latest
```

2. Create an Amazon ECR repository for your image.

```
PS C:\> aws ecr create-repository --repository-name iis
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "iis",
        "repositoryArn": "arn:aws:ecr:region:aws_account_id:repository/iis",
        "createdAt": 1481845593.0,
        "repositoryUri": "aws_account_id.dkr.ecr.region.amazonaws.com/iis"
    }
}
```

3. Tag the image with the `repositoryUri` that was returned from the previous command.

```
PS C:\> docker tag microsoft/iis aws_account_id.dkr.ecr.region.amazonaws.com/iis
```

4. To authenticate Docker to an Amazon ECR registry with get-login-password, run the `aws ecr get-login-password` command. When passing the authentication token to the `docker login` command, you specify the AWS username and your Amazon ECR registry URI.

Important

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

```
PS C:\> aws ecr get-login-password --region region | docker login --username AWS --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com/iis
```

5. Push the image to Amazon ECR.

```
PS C:\> docker push aws_account_id.dkr.ecr.region.amazonaws.com/iis
The push refers to a repository [111122223333.dkr.ecr.us-west-2.amazonaws.com/iis]
1e4f77a75bd4: Pushed
ac90fb7da567: Pushed
c7090349c7b3: Pushed
b9454c3094c6: Skipped foreign layer
3fd27ecef6a3: Skipped foreign layer
latest: digest: sha256:0ddc7af8691072bb2dd8b3f189388b33604c90774d3dc0485b1bf379f9bec4c5
size: 1574
```

Building your own Amazon ECS-optimized Windows AMI

EC2 Image Builder can be used to build your own custom Amazon ECS-optimized Windows AMI. This makes it easy to use a Windows AMI with your own license on Amazon ECS. Amazon ECS provides a managed Image Builder component which provides the system configuration needed to run Windows instances to host your containers. Each Amazon ECS managed component includes a specific container agent and Docker version. You can customize your image to use either the latest Amazon ECS managed component, or if an older container agent or Docker version is needed you can specify a different component.

For a full walkthrough of using EC2 Image Builder, see [Getting started with EC2 Image Builder](#) in the *EC2 Image Builder User Guide*.

When building your own Amazon ECS-optimized Windows AMI using EC2 Image Builder, you create an image recipe. Your image recipe must meet the following requirements:

- The **Source image** should be based on Windows Server 20H2 Core, Windows Server 2004 Core, Windows Server 2016 Full, Windows Server 2019 Core, Windows Server 2019 Full, Windows Server 2022 Core, or Windows Server 2022 Full. Any other Windows operating system is not supported and may not be compatible with the component.
- When specifying the **Build components**, the `ecs-optimized-ami-windows` component is required. The `update-windows` component is recommended, which ensures the image contains the latest security updates.

Selected components (2)
Expand the component to view versioning options. To sort the build sequence, drag the components up and down.

Sequence	Component (drag the component up or down to change the sequence)	Owner	Expand versioning
1	ecs-optimized-ami-windows Versioning options <ul style="list-style-type: none"> <input checked="" type="radio"/> Use latest available component version <input type="radio"/> Specify component version 	Owner: Amazon	X
2	update-windows Versioning options	Owner: Amazon	X

To specify a different component version, expand the **Versioning options** menu and specify the component version you want to use. For more information, see [Listing the ecs-optimized-ami-windows component versions \(p. 849\)](#).

Listing the ecs-optimized-ami-windows component versions

When creating an EC2 Image Builder recipe and specifying the `ecs-optimized-ami-windows` component, you can either use the default option or you can specify a specific component version. To determine what component versions are available, along with the Amazon ECS container agent and Docker versions contained within the component, you can use the AWS Management Console.

To list the available `ecs-optimized-ami-windows` component versions

1. Open the EC2 Image Builder console at <https://console.aws.amazon.com/imagebuilder/>.
2. On the navigation bar, select the Region that are building your image in.
3. In the navigation pane, under the **Saved configurations** menu, choose **Components**.
4. On the **Components** page, in the search bar type `ecs-optimized-ami-windows` and pull down the qualification menu and select **Quick start (Amazon-managed)**.
5. Use the **Description** column to determine the component version with the Amazon ECS container agent and Docker version your image requires.

Using gMSAs for Windows Containers

Amazon ECS supports Active Directory authentication for Windows containers through a special kind of service account called a *group Managed Service Account* (gMSA).

Windows based network applications such as .NET applications often use Active Directory to facilitate authentication and authorization management between users and services. Developers commonly design their applications to integrate with Active Directory and run on domain-joined servers for this purpose. Because Windows containers cannot be domain-joined, you must configure a Windows container to run with gMSA.

A Windows container running with gMSA relies on its host Amazon EC2 instance to retrieve the gMSA credentials from the Active Directory domain controller and provide them to the container instance. For more information, see [Create gMSAs for Windows containers](#).

Note

This feature is not supported on Windows containers on Fargate.

Topics

- [Considerations \(p. 850\)](#)
- [Prerequisites \(p. 850\)](#)
- [Setting Up gMSA-capable Windows Containers on Amazon ECS \(p. 850\)](#)

Considerations

The following should be considered when using gMSAs for Windows containers:

- When using the Amazon ECS-optimized Windows Server 2016 Full AMI for your container instances, the container hostname must be the same as the gMSA account name defined in the credential spec file. To specify a hostname for a container, use the `hostname` container definition parameter. For more information, see [Network settings \(p. 223\)](#).

Prerequisites

The following are prerequisites for using the gMSA for Windows containers feature with Amazon ECS.

- An Active Directory that your Amazon ECS Windows container instances can join. Amazon ECS supports the following:
 - AWS Directory Service, which is an AWS managed Active Directory hosted on Amazon EC2. For more information, see [Getting Started with AWS Managed Microsoft AD](#) in the *AWS Directory Service Administration Guide*.
 - On-premises Active Directory, as long as the Amazon ECS Windows container instance can join the domain. For more information, see [AWS Direct Connect](#).
- An existing gMSA account in the Active Directory. For more information, see [Create gMSAs for Windows containers](#).
- The Amazon ECS Windows container instance hosting the Amazon ECS task must be domain joined to the Active Directory and be a member of the Active Directory security group that has access to the gMSA account.

Setting Up gMSA-capable Windows Containers on Amazon ECS

Amazon ECS uses a credential spec file that contains the gMSA metadata used to propagate the gMSA account context to the Windows container. You can generate the credential spec file and reference it in the `dockerSecurityOptions` field in your task definition. The credential spec file does not contain any secrets.

The following is an example credential spec file:

```
{  
    "CmsPlugins": [  
        "ActiveDirectory"  
    ],  
    "DomainJoinConfig": {  
        "Sid": "S-1-5-21-2554468230-2647958158-2204241789",  
        "MachineAccountName": "WebApp01",  
        "Guid": "8665abd4-e947-4dd0-9a51-f8254943c90b",  
        "DnsTreeName": "example.com",  
        "DnsName": "example.com",  
        "NetBiosName": "example"  
    },  
    "ActiveDirectoryConfig": {  
        "GroupManagedServiceAccounts": [  
            {  
                "Name": "WebApp01",  
                "Scope": "example.com"  
            }  
        ]  
    }  
}
```

Referencing a Credential Spec File in a Task Definition

Amazon ECS supports the following ways to reference the credential spec file in the `dockerSecurityOptions` field of a task definition.

Topics

- [Amazon S3 Bucket \(p. 851\)](#)
- [SSM Parameter Store parameter \(p. 852\)](#)
- [Local File \(p. 852\)](#)

Amazon S3 Bucket

Add the credential spec to an Amazon S3 bucket and then reference the Amazon Resource Name (ARN) of the Amazon S3 bucket in the `dockerSecurityOptions` field of the task definition.

```
{  
    "family": "",  
    "executionRoleArn": "",  
    "containerDefinitions": [  
        {  
            "name": "",  
            ...  
            "dockerSecurityOptions": [  
                "credentialspec:arn:aws:s3:::${BucketName}/${ObjectName}"  
            ],  
            ...  
        },  
        ...  
    ]  
}
```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the Amazon S3 bucket.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "VisualEditor",
        "Effect": "Allow",
        "Action": [
            "s3:Get*",
            "s3>List*"
        ],
        "Resource": [
            "arn:aws:s3:::{bucket_name}",
            "arn:aws:s3:::{bucket_name}/{object}"
        ]
    }
]
```

SSM Parameter Store parameter

Add the credential spec to an SSM Parameter Store parameter and then reference the Amazon Resource Name (ARN) of the SSM Parameter Store parameter in the `dockerSecurityOptions` field of the task definition.

```
{
    "family": "",
    "executionRoleArn": "",
    "containerDefinitions": [
        {
            "name": "",
            ...
            "dockerSecurityOptions": [
                "credentialspec:arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ],
            ...
        },
        ...
    ],
    ...
}
```

You must also add the following permissions as an inline policy to the Amazon ECS task execution IAM role to give your tasks access to the SSM Parameter Store parameter.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:region:111122223333:parameter/parameter_name"
            ]
        }
    ]
}
```

Local File

With the credential spec details in a local file, reference the file path in the `dockerSecurityOptions` field of the task definition.

```
{  
    "family": "",  
    "executionRoleArn": "",  
    "containerDefinitions": [  
        {  
            "name": "",  
            ...  
            "dockerSecurityOptions": [  
                "credentialspec:file://CredentialSpecFile.json"  
            ],  
            ...  
        },  
        ...  
    ]  
}
```

Document history

The following table describes the major updates and new features for the *Amazon Elastic Container Service Developer Guide*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
Amazon ECS-optimized Linux AMI build script	Amazon ECS has open-sourced the build scripts that are used to build the Linux variants of the Amazon ECS-optimized AMI. For more information, see Amazon ECS-optimized Linux AMI build script (p. 361) .	19 Nov 2021
Container instance health	Amazon ECS adds support for container instance health monitoring. For more information, see Container instance health (p. 647) .	10 Nov 2021
Support for Windows Amazon ECS Exec	Amazon ECS Exec supports Windows. For more information, see Using Amazon ECS Exec for debugging (p. 805) .	01 Nov 2021
Support for Windows Amazon ECS Exec	Container instance health monitoring For more information, see Container instance health (p. 647) .	01 Nov 2021
Support for Windows containers on Fargate.	Amazon ECS supports Windows containers on Fargate. For more information, see Windows platform versions (p. 173) .	28 Oct 2021
GPU support for external instances on Amazon ECS Anywhere	Amazon ECS supports specifying GPU requirements in the task definition for tasks run on external instances. For more information, see Working with GPUs on Amazon ECS (p. 250) and Registering an external instance to a cluster (p. 418) .	8 Oct 2021
Support of awsvpc network mode on Windows	Amazon ECS supports awsvpc network mode on Windows. For more information, see Task networking with the awsvpc network mode (p. 278) .	15 July 2021
General availability of Bottlerocket	Amazon ECS supports an Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI. For more information, see Using Bottlerocket with Amazon ECS (p. 362) .	30 June 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 Scheduled tasks (p. 524) .	25 June 2021
AWS managed policies for Amazon ECS	Amazon ECS added documentation of AWS managed policies for service-linked roles. For more information, see AWS managed policies for Amazon Elastic Container Service (p. 674) .	08 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 Getting started with Amazon ECS using the AWS CDK (p. 23) .	27 May 2021

Change	Description	Date
Amazon ECS Anywhere	Amazon ECS has added support for registering an on-premise server or virtual machine (VM) with your cluster. For more information, see External instances (Amazon ECS Anywhere) (p. 414) .	25 May 2021
Amazon ECS-optimized Windows Server 20H2 Core AMI	Amazon ECS has added support for a new Windows Amazon ECS-optimized AMI variant for Windows Server 20H2 Core. For more information, see Amazon ECS-optimized AMI (p. 333) .	19 April 2021
Amazon ECS Exec	Amazon ECS has released a new debugging tool called ECS Exec. For more information, see Using Amazon ECS Exec for debugging (p. 805) .	15 March 2021
VPC endpoint policy support	Amazon ECS now supports VPC endpoint policies. For more information, see Creating a VPC endpoint policy for Amazon ECS (p. 716) .	11 Jan 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 Creating a service using the new console (p. 547) and Run a standalone task (p. 510) .	28 December 2020
Capacity provider update	Amazon ECS added support for updating an existing Auto Scaling group capacity provider. For more information, see Updating an Auto Scaling group capacity provider (p. 186) .	23 November 2020
ECS now supporting Amazon FSx for Windows File Server for Windows tasks	Amazon ECS added support for specifying Amazon FSx for Windows File Server volumes in Windows task definitions. For more information, see FSx for Windows File Server volumes (p. 260) .	11 November 2020
VPC dual-stack mode support added	Amazon ECS added support for using a VPC in dual-stack mode with tasks using the <code>awsvpc</code> network mode, which provides support for IPv6 addresses. For more information, see Using a VPC in dual-stack mode (p. 282) .	5 November 2020
Task metadata endpoint v4 update	Amazon ECS added additional metadata to the task metadata endpoint v4 output. For more information, see Task metadata endpoint version 4 (p. 477) .	5 November 2020
Support for Local Zones and Wavelength Zones	Amazon ECS added support for workloads in Local Zones and Wavelength Zones. For more information, see Amazon ECS clusters in Local Zones, Wavelength Zones, and AWS Outposts (p. 191) .	4 September 2020
Amazon ECS variant of Bottlerocket AMI	Bottlerocket is a Linux-based open source operating system that is purpose-built by AWS for running containers. An Amazon ECS-optimized AMI variant of the Bottlerocket operating system is provided as an AMI you can use when launching Amazon ECS container instances. For more information, see Using Bottlerocket with Amazon ECS (p. 362) .	31 August 2020

Change	Description	Date
Task metadata endpoint version 4 updated for network rate stats	The task metadata endpoint version 4 has been updated to provide network rate stats for Amazon ECS tasks that use the <code>awsvpc</code> or <code>bridge</code> network modes hosted on Amazon EC2 instances running at least version 1.43.0 of the container agent. For more information, see Task metadata endpoint version 4 (p. 477) .	10 August 2020
Fargate usage metrics	AWS Fargate provides CloudWatch usage metrics which provide visibility into your accounts usage of Fargate On-Demand and Fargate Spot resources. For more information, see Usage metrics (p. 167) .	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 Using the AWS Copilot command line interface (p. 49) .	9 July 2020
AWS Fargate platform versions deprecation schedule	The Fargate platform version deprecation schedule has been added. For more information, see AWS Fargate platform version deprecation (p. 173) .	8 July 2020
AWS Fargate Region expansion	Amazon ECS on AWS Fargate has expanded to the Europe (Milan) Region.	25 June 2020
Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI released	Amazon ECS released an Amazon ECS-optimized Amazon Linux 2 (Inferentia) AMI for inferential workloads. For more information, see Amazon ECS-optimized AMI (p. 333) .	24 June 2020
Added support for deleting capacity providers	Amazon ECS added support for deleting Auto Scaling group capacity providers. For more information, see Deleting an Auto Scaling group capacity provider (p. 188) .	11 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 Fargate task storage (p. 256) .	28 May 2020
Environment variable file support	Added support for specifying environment variable files in a task definition, which enables you to bulk add environment variables to your containers. For more information, see Specifying environment variables (p. 315) .	18 May 2020
AWS Fargate Region expansion	AWS Fargate with Amazon ECS has expanded to the Africa (Cape Town) Region.	11 May 2020

Change	Description	Date
Service quota updated	<p>The following service quota was updated:</p> <ul style="list-style-type: none">• Clusters per account was raised from 2,000 to 10,000. <p>For more information, see Amazon ECS service quotas (p. 611).</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"> • Added support for using Amazon EFS file system volumes for persistent task storage. For more information, see Amazon EFS volumes (p. 257). • The ephemeral task storage has been increased to 20 GB. For more information, see Fargate task storage (p. 256). • The network traffic behavior to and from tasks has been updated. Starting with platform version 1.4, all Fargate tasks receive a single elastic network interface (referred to as the task ENI) and all network traffic flows through that ENI within your VPC and will be visible to you through your VPC flow logs. For more information, see Fargate Task Networking in the <i>Amazon Elastic Container Service User Guide for AWS Fargate</i>. • 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. 645). • Added support for the task metadata endpoint v4 which provides additional information for your Fargate tasks, including network stats for the task and which Availability Zone the task is running in. For more information, see Task metadata endpoint version 4 (p. 477). • Added support for the <code>SYS_PTRACE</code> Linux parameter in container definitions. For more information, see Linux parameters (p. 232). • 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. 	8 April 2020

Change	Description	Date
	For more information, see AWS Fargate platform versions (p. 169) .	
Amazon EFS file system support for task volumes	Amazon EFS file systems can be used as data volumes for both your Amazon ECS and Fargate tasks. For more information, see Amazon EFS volumes (p. 257) .	8 April 2020
Amazon ECS Task Metadata Endpoint version 4	Beginning with Amazon ECS container agent version 1.39.0 and Fargate platform version 1.4.0, an environment variable named <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 Docker stats are available to tasks. For more information, see Task metadata endpoint version 4 (p. 477) .	8 April 2020
Support for specific versions of Secrets Manager secrets to be injected as environment variables	Added support for specifying sensitive data using specific versions of Secrets Manager secrets. For more information, see Injecting sensitive data as an environment variable (p. 305) .	24 Feb 2020
Added additional CodeDeploy deployment configuration options for blue/green deployments	The CodeDeploy service added new canary and linear deployment configurations for the Amazon ECS deployment type. The ability to define custom deployment configurations is also available. For more information, see Blue/Green deployment with CodeDeploy (p. 565) .	6 Feb 2020
Added the <code>efsVolumeConfiguration</code> task definition parameter	The <code>efsVolumeConfiguration</code> task definition parameter is in public preview, which makes it easier to use Amazon EFS file systems with your Amazon ECS tasks. For more information, see Amazon EFS volumes (p. 257) .	17 Jan 2020
Amazon ECS container agent logging behavior updated	The Amazon ECS container agent logging locations and rotation behavior has been updated. For more information, see Amazon ECS Container Agent Log (p. 829) .	13 Jan 2020
Fargate Spot	Amazon ECS added support for running tasks using Fargate Spot. For more information, see AWS Fargate capacity providers (p. 180) .	3 Dec 2019
Cluster Auto Scaling	Amazon ECS cluster auto scaling enables you to have more control over how you scale tasks within a cluster. For more information, see Amazon ECS cluster auto scaling (p. 189) .	3 Dec 2019
Cluster Capacity Providers	Amazon ECS cluster capacity providers determine the infrastructure to use for your tasks. For more information, see Amazon ECS capacity providers (p. 178) .	3 Dec 2019
Creating a cluster on an AWS Outposts	Amazon ECS now supports creating clusters on an AWS Outposts. For more information, see the section called "Amazon Elastic Container Service on AWS Outposts" (p. 722).	3 Dec 2019

Change	Description	Date
Service Action Events	Amazon ECS now sends events to Amazon EventBridge when certain service actions occur. For more information, see Service action events (p. 638) .	25 Nov 2019
Amazon ECS GPU-optimized AMI Supports G4 Instances	Amazon ECS added support for the g4 instance type family when using the Amazon ECS GPU-optimized AMI. For more information, see Working with GPUs on Amazon ECS (p. 250) .	8 Oct 2019
Amazon ECS CLI v1.17.0	New version of the Amazon ECS CLI released. This release added support for specifying a FireLens configuration using the ECS Parameters file. For more information, see Using Amazon ECS Parameters (p. 155) .	2 Oct 2019
FireLens for Amazon ECS	FireLens for Amazon ECS is in general availability. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Custom log routing (p. 289) .	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
Deep Learning Containers with Elastic Inference on Amazon ECS	Amazon ECS supports attaching Amazon Elastic Inference accelerators to your containers to make running deep learning inference workloads more efficient. For more information, see Deep Learning Containers with Elastic Inference on Amazon ECS (p. 726) .	3 Sept 2019
FireLens for Amazon ECS	FireLens for Amazon ECS is in public preview. FireLens for Amazon ECS enables you to use task definition parameters to route logs to an AWS service or partner destination for log storage and analytics. For more information, see Custom log routing (p. 289) .	30 Aug 2019
CloudWatch Container Insights	CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Amazon ECS CloudWatch Container Insights (p. 645) .	30 Aug 2019
Container Level Swap Configuration	Amazon ECS added support for controlling the usage of swap memory space on your Linux container instances at the container level. Using a per-container swap configuration, each container within a task definition can have swap enabled or disabled, and for those that have it enabled, the maximum amount of swap space used can be limited. For more information, see Managing container swap space (p. 277) .	16 Aug 2019
AWS Fargate region expansion	AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Hong Kong) Region.	06 Aug 2019
Elastic Network Interface Trunking	Added additional supported Amazon EC2 instance types for ENI trunking feature. For more information, see Supported Amazon EC2 instance types (p. 375) .	1 Aug 2019

Change	Description	Date
Registering Multiple Target Groups with a Service	Added support for specifying multiple target groups in a service definition. For more information, see Registering multiple target groups with a service (p. 589) .	30 July 2019
Specifying Sensitive Data Using Secrets Manager Secrets	Added tutorial for specifying sensitive data using Secrets Manager secrets. For more information, see Tutorial: Specifying sensitive data using Secrets Manager secrets (p. 764) .	20 July 2019
Amazon ECS CLI v1.15.0	New version of the Amazon ECS CLI released. For more information, see Amazon ECS CLI Changelog .	9 July 2019
CloudWatch Container Insights	Amazon ECS has added support for CloudWatch Container Insights. For more information, see Amazon ECS CloudWatch Container Insights (p. 645) .	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 How Amazon Elastic Container Service works with IAM (p. 658) .	27 June 2019
New Amazon ECS-optimized AMI patched for AWS-2019-005	Amazon ECS has updated the Amazon ECS-optimized AMI to address the vulnerabilities described in AWS-2019-005 .	17 June 2019
Elastic Network Interface Trunking	Amazon ECS introduces support for launching container instances using supported Amazon EC2 instance types that have increased elastic network interface (ENI) density. Using these instance types and opting in to the <code>awsvpcTrunking</code> account setting provides increased ENI density on newly launched container instances which allows you to place more tasks on each container instance. For more information, see Elastic network interface trunking (p. 372) .	6 June 2019
AWS Fargate platform version 1.3.0 update	Beginning on May 1, 2019, any new Fargate task that is launched supports the <code>splunk</code> log driver in addition to the <code>awslogs</code> log driver. For more information, see Storage and logging (p. 225) .	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 Specifying sensitive data (p. 303) .	1 May 2019
AWS Fargate platform version 1.3.0 update	Beginning on April 2, 2019, any new Fargate task that is launched supports injecting sensitive data into your containers by storing your sensitive data in either AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters and then referencing them in your container definition. For more information, see Specifying sensitive data (p. 303) .	2 Apr 2019

Change	Description	Date
AWS Fargate platform version 1.3.0 update	Beginning on March 27, 2019, any new Fargate task launched can use additional task definition parameters that enable you to define a proxy configuration, dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Proxy configuration (p. 240) , Container dependency (p. 236) , and Container timeouts (p. 237) .	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 External deployment (p. 569) .	27 Mar 2019
AWS Deep Learning Containers on Amazon ECS	AWS Deep Learning Containers are a set of Docker images for training and serving models in TensorFlow on Amazon Elastic Container Service (Amazon ECS). Deep Learning Containers provide optimized environments with TensorFlow, Nvidia CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries and are available in Amazon ECR. For more information, see AWS Deep Learning Containers on Amazon ECS (p. 726) .	27 Mar 2019
Amazon ECS introduces enhanced container dependency management	Amazon ECS introduces additional task definition parameters that enable you to define dependencies for container startup and shutdown as well as a per-container start and stop timeout value. For more information, see Container dependency (p. 236) .	7 Mar 2019
Amazon ECS CLI v1.13.0	New version of the Amazon ECS CLI released. For more information, see Amazon ECS CLI Changelog .	7 Mar 2019
Amazon ECS introduces the <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 Amazon Resource Names (ARNs) and IDs (p. 325) .	8 Feb 2019
Amazon ECS supports GPU workloads	Amazon ECS introduces support for GPU workloads by enabling you to create clusters with GPU-enabled container instances. In a task definition you can specify the number of required GPUs and the ECS agent will pin the physical GPUs to the container. For more information, see Working with GPUs on Amazon ECS (p. 250) .	4 Feb 2019
Amazon ECS expanded secrets support	Amazon ECS expanded support for using AWS Secrets Manager secrets directly in your task definitions to inject sensitive data into your containers. For more information, see Specifying sensitive data (p. 303) .	21 Jan 2019

Change	Description	Date
Interface VPC Endpoints (AWS PrivateLink)	<p>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.</p> <p>For more information, see Interface VPC Endpoints (AWS PrivateLink).</p>	26 Dec 2018
AWS Fargate platform version 1.3.0	<p>New AWS Fargate platform version released, which contains:</p> <ul style="list-style-type: none"> Added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers. <p>For more information, see Specifying sensitive data (p. 303).</p> <ul style="list-style-type: none"> Added task recycling for Fargate tasks, which is the process of refreshing tasks that are a part of an Amazon ECS service. <p>For more information, see Task maintenance in the <i>Amazon Elastic Container Service User Guide for AWS Fargate</i>.</p> <p>For more information, see AWS Fargate platform versions (p. 169).</p>	17 Dec 2018
Service limits updated	<p>The following service limits were updated:</p> <ul style="list-style-type: none"> Number of clusters per Region, per account was raised from 1000 to 2000. Number of container instances per cluster was raised from 1000 to 2000. Number of services per cluster was raised from 500 to 1000. <p>For more information, see Amazon ECS service quotas (p. 611).</p>	14 Dec 2018
AWS Fargate region expansion	<p>AWS Fargate with Amazon ECS has expanded to the Asia Pacific (Mumbai) and Canada (Central) Regions.</p> <p>For more information, see AWS Fargate platform versions (p. 169).</p>	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 Blue/Green deployment with CodeDeploy (p. 565).</p>	27 Nov 2018

Change	Description	Date
Amazon ECS-optimized Amazon Linux 2 (arm64) AMI released	<p>Amazon ECS released an Amazon ECS-optimized Amazon Linux 2 AMIs for arm64 architecture.</p> <p>For more information, see Amazon ECS-optimized AMI (p. 333).</p>	26 Nov 2018
Amazon ECS CLI v1.11.2	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> • Added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers. For more information, see Using Amazon ECS Parameters (p. 155). • Added support for specifying the <code>ipcMode</code> and <code>pidMode</code> Docker flags in task definitions. For more information, see Using Amazon ECS Parameters (p. 155). 	19 Nov 2018
Added support for additional Docker flags in task definitions	<p>Amazon ECS introduced support for the following Docker flags in task definitions:</p> <ul style="list-style-type: none"> • IPC mode (p. 246) • PID mode (p. 247) 	16 Nov 2018
Amazon ECS secrets support	<p>Amazon ECS added support for using AWS Systems Manager Parameter Store parameters to inject sensitive data into your containers.</p> <p>For more information, see Specifying sensitive data (p. 303).</p>	15 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 Resources and tags (p. 604).</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 Amazon ECS on AWS Fargate (p. 161).</p>	07 Nov 2018
Service limits updated	<p>The following service limits were updated:</p> <ul style="list-style-type: none"> • Number of tasks using the Fargate launch type, per Region, per account was raised from 20 to 50. • Number of public IP addresses for tasks using the Fargate launch type was raised from 20 to 50. <p>For more information, see Amazon ECS service quotas (p. 611).</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 Amazon ECS on AWS Fargate (p. 161).</p>	26 Oct 2018
Amazon ECS CLI v1.10.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> • Added the ecs-cli registry-creds command, which facilitates the creation and use of private registry credentials within Amazon ECS. For more information, see ecs-cli registry-creds (p. 140). • Added support for Amazon Linux 2. For more information, see Amazon ECS-optimized AMI (p. 333). 	25 October 2018
Amazon ECS-optimized Amazon Linux 2 AMI Released	<p>Amazon ECS vends Linux AMIs that are optimized for the service in two variants. The latest and recommended version is based on x; Amazon ECS also vends AMIs that are based on the Amazon Linux AMI, but we recommend that you migrate your workloads to the Amazon Linux 2 variant, as support for the Amazon Linux AMI will end no later than June 30, 2020.</p> <p>For more information, see Amazon ECS-optimized AMI (p. 333).</p>	18 October 2018
Amazon ECS CLI v1.9.0	<p>New version of the Amazon ECS CLI released, which added the following functionality:</p> <ul style="list-style-type: none"> • Added support for service discovery. For more information, see Tutorial: Creating an Amazon ECS Service That Uses Service Discovery Using the Amazon ECS CLI (p. 70). • Added support for Amazon EC2 Spot instances in Amazon ECS clusters. • Added support for custom user data. 	18 October 2018
Amazon ECS Task Metadata Endpoint version 3	<p>Beginning with version 1.21.0 of the Amazon ECS container agent, the agent injects an environment variable called <code>ECS_CONTAINER_METADATA_URI</code> into each container in a task. When you query the task metadata version 3 endpoint, various task metadata and Docker stats are available to tasks that use the <code>awsvpc</code> network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see Amazon ECS task metadata endpoint (p. 476).</p>	18 October 2018
Amazon ECS service discovery Region expansion	<p>Amazon ECS service discovery has expanded support to the Canada (Central), South America (São Paulo), Asia Pacific (Seoul), Asia Pacific (Mumbai), and Europe (Paris) Regions.</p> <p>For more information, see Service Discovery (p. 599).</p>	27 September 2018

Change	Description	Date
Added support for additional Docker flags in container definitions	<p>Amazon ECS introduced support for the following Docker flags in container definitions:</p> <ul style="list-style-type: none"> • System controls (p. 238) • Interactive (p. 239) • Pseudo terminal (p. 239) 	17 Sept 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 Private registry authentication for tasks (p. 300).</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"> • Added support for Docker volumes in Docker compose files. For more information, see ecs-cli compose (p. 102). • Added support for task placement constraints and strategies in Docker compose files. For more information, see ecs-cli compose (p. 102). • Added support for private registry authentication in Docker compose files. For more information, see ecs-cli compose (p. 102). • Added support for --force-update on compose up to force relaunching of tasks. For more information, see ecs-cli compose up (p. 111). 	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 Service Discovery (p. 599).</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 Scheduled tasks (p. 524).</p>	28 August 2018
Private registry authentication using AWS Secrets Manager support	<p>Amazon ECS introduced support for private registry authentication using AWS Secrets Manager. This feature enables you to store your credentials securely and then reference them in your container definition, which allows your tasks to use private images.</p> <p>For more information, see Private registry authentication for tasks (p. 300).</p>	16 August 2018

Change	Description	Date
Docker volume support added	<p>Amazon ECS introduced support for Docker volumes.</p> <p>For more information, see Using data volumes in tasks (p. 256).</p>	9 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 Amazon ECS on AWS Fargate (p. 161).</p>	19 July 2018
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"> • Added support for container <code>healthcheck</code> and <code>devices</code> in Docker compose files. For more information, see ecs-cli compose (p. 102). 	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"> • 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. 533). • DAEMON—The daemon scheduling strategy deploys exactly one task on each active container instance that meets all of the task placement constraints that you specify in your cluster. The service scheduler evaluates the task placement constraints for running tasks and will stop tasks that do not meet the placement constraints. When using this strategy, there is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies. For more information, see Daemon (p. 532). <p>Note Fargate tasks do not support the DAEMON scheduling strategy.</p> <p>For more information, see Service scheduler concepts (p. 531).</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"> • Added support for Docker compose file syntax version 3. For more information, see ecs-cli compose (p. 102). 	5 June 2018

Change	Description	Date
Amazon ECS container agent v1.18.0	<p>New version of the Amazon ECS container agent released, which added the following functionality:</p> <ul style="list-style-type: none"> • Added procedure to manually install the container agent from a S3 URL on non-Amazon Linux EC2 instance, including a PGP signature method for verifying the Amazon ECS container agent installation file. For more information, see Installing the Amazon ECS container agent (p. 431). • Added procedure to manually install the container agent from a S3 URL on a Windows EC2 instance, including a PGP signature method for verifying the Amazon ECS container agent installation file. For more information, see Getting started with the Amazon ECS console using Amazon EC2 Windows containers (p. 41). • Added support for customizing the container agent image pull behavior using the <code>ECS_IMAGE_PULL_BEHAVIOR</code> parameter. For more information, see Amazon ECS container agent configuration (p. 454). <p>For more information, see amazon-ecs-agent github.</p>	24 May 2018
Added Support for bridge and host Network Modes When Configuring Service Discovery	Added support for configuring service discovery for Amazon ECS services using task definitions that specify the bridge or host network modes. For more information, see Service Discovery (p. 599) .	22 May 2018
Added support for additional Amazon ECS-optimized AMI metadata parameters	Added subparameters that allow you to programmatically retrieve the Amazon ECS-optimized AMI ID, image name, operating system, container agent version, and runtime version. Query the metadata using the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI metadata (p. 339) .	9 May 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 Amazon ECS on AWS Fargate (p. 161).</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"> • Added support for the ECS CLI to automatically retrieve the latest stable Amazon ECS-optimized AMI by querying the Systems Manager Parameter Store API during the cluster resource creation process. This requires the user account that you are using to have the required Systems Manager permissions. For more information, see ecs-cli up (p. 81). • Added support for the <code>shm_size</code> and <code>tmpfs</code> parameters in compose files. For more information, see ecs-cli compose (p. 102). <p>For more information about the updated ECS CLI syntax, see Amazon ECS command line reference (p. 73).</p>	19 April 2018
Amazon ECS-optimized AMI Metadata Retrieval	Added ability to programmatically retrieve Amazon ECS-optimized AMI metadata using the Systems Manager Parameter Store API. For more information, see Retrieving Amazon ECS-Optimized AMI metadata (p. 339) .	10 April 2018
Amazon ECS CLI download verification	Added new PGP signature method for verifying the Amazon ECS CLI installation file. For more information, see Installing the Amazon ECS CLI (p. 55) .	5 April 2018
AWS Fargate Platform Version	<p>New AWS Fargate platform version released, which contains:</p> <ul style="list-style-type: none"> • Added support for Amazon ECS task metadata endpoint (p. 476). • Added support for Health check (p. 218). • Added support for Service Discovery (p. 599) <p>For more information, see AWS Fargate platform versions (p. 169).</p>	26 March 2018
Amazon ECS Service Discovery	Added integration with Route 53 to support Amazon ECS service discovery. For more information, see Service Discovery (p. 599) .	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"> • Updated the AMI to <code>amzn-ami-2017.09.k-amazon-ecs-optimized</code>. <p>For more information about the updated ECS CLI syntax, see Amazon ECS command line reference (p. 73).</p>	20 March 2018

Change	Description	Date
Docker shm-size and tmpfs support	<p>Added support for the Docker shm-size and tmpfs parameters in Amazon ECS task definitions.</p> <p>For more information about the updated ECS CLI syntax, see Linux parameters (p. 232).</p>	20 March 2018
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"> • Added support for the us-gov-west-1 Region. • Added --force-deployment flag for the compose service command. For more information, see ecs-cli compose service (p. 114). • Added support for aws_session_token in ECS profiles. For more information, see ecs-cli configure profile (p. 78). • Updated the AMI to amzn-ami-2017.09.j-amazon-ecs-optimized. <p>For more information about the updated ECS CLI syntax, see Amazon ECS command line reference (p. 73).</p>	09 March 2018
Container Health Checks	Added support for Docker health checks in container definitions. For more information, see Health check (p. 218) .	08 March 2018
AWS Fargate	Added overview for Amazon ECS with AWS Fargate. For more information, see Amazon ECS on AWS Fargate (p. 161) .	22 February 2018
Amazon ECS Task Metadata Endpoint	Beginning with version 1.17.0 of the Amazon ECS container agent, various task metadata and Docker stats are available to tasks that use the awsvpc network mode at an HTTP endpoint that is provided by the Amazon ECS container agent. For more information, see Amazon ECS task metadata endpoint (p. 476) .	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 Target tracking scaling policies (p. 593).</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

Change	Description	Date
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"> • Ability to create empty clusters with the up command. • Added --health-check-grace-period flag for the compose service up command. • Updated the AMI to amzn-ami-2017.09.g-amazon-ecs-optimized. <p>For more information about the updated ECS CLI syntax, see Amazon ECS command line reference (p. 73).</p>	19 January 2018
Docker 17.09 support	Added support for Docker 17.09. For more information, see Amazon ECS-optimized AMI (p. 333) .	18 January 2018
Elastic Load Balancing health check initialization wait period	Added ability to specify a wait period for health checks.	27 December 2017
New service scheduler behavior	Updated information about the behavior for service tasks that fail to launch. Documented new service event message that triggers when a service task has consecutive failures. For more information about this updated behavior, see Additional service concepts (p. 533) .	11 January 2018
Task-level CPU and memory	Added support for specifying CPU and memory at the task-level in task definitions. For more information, see TaskDefinition .	12 December 2017
Task execution role	<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"> • Calls to Amazon ECR to pull the container image • Calls to CloudWatch to store container application logs <p>For more information, see Amazon ECS task execution IAM role (p. 691).</p>	7 December 2017
Windows containers support GA	Added support for Windows Server 2016 containers. For more information, see Amazon EC2 Windows containers (p. 842) .	5 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"> • Support for task networking • Support for AWS Fargate • Support for viewing CloudWatch Logs data from a task <p>For more information, see ECS CLI changelog.</p>	29 November 2017

Change	Description	Date
AWS Fargate GA	Added support for launching Amazon ECS services using the Fargate launch type. For more information, see Amazon ECS launch types (p. 247) .	29 November 2017
Amazon ECS name change	Amazon Elastic Container Service is renamed (previously Amazon EC2 Container Service).	21 November 2017
Task networking	The task networking features provided by the awsvpc network mode give Amazon ECS tasks the same networking properties as Amazon EC2 instances. When you use the awsvpc network mode in your task definitions, every task that is launched from that task definition gets its own elastic network interface, a primary private IP address, and an internal DNS hostname. The task networking feature simplifies container networking and gives you more control over how containerized applications communicate with each other and other services within your VPCs. For more information, see Amazon ECS task networking (p. 278) .	14 November 2017
Amazon ECS CLI v1.0.0	New version of the Amazon ECS CLI released, which added the following features: <ul style="list-style-type: none"> Support for adding multiple named profiles and cluster configurations Support for custom task definition parameters specified using <code>--ecs-params</code> Support for running the Amazon ECS CLI on Windows For more information, see ECS CLI changelog .	7 November 2017
Amazon ECS container metadata	Amazon ECS containers are now able to access metadata such as their Docker container or image ID, networking configuration, or Amazon ARNs. For more information, see Amazon ECS container metadata file (p. 472) .	2 November 2017
Docker 17.06 support	Added support for Docker 17.06. For more information, see Amazon ECS-optimized AMI (p. 333) .	2 November 2017
Support for Docker flags: device and init	Added support for Docker's device and init features in task definitions using the <code>LinuxParameters</code> parameter (<code>devices</code> and <code>initProcessEnabled</code>). For more information, see LinuxParameters .	2 November 2017
Support for Docker flags: cap-add and cap-drop	Added support for Docker's cap-add and cap-drop features in task definitions using the <code>LinuxParameters</code> parameter (<code>capabilities</code>). For more information, see LinuxParameters .	22 September 2017
Network Load Balancer support	Amazon ECS added support for Network Load Balancers in the Amazon ECS console. For more information, see Creating a Network Load Balancer (p. 584) .	7 September 2017

Change	Description	Date
RunTask overrides	Added support for task definition overrides when running a task. This allows you to run a task while changing a task definition without the need to create a new task definition revision. For more information, see Run a standalone task (p. 510) .	27 June 2017
Amazon ECS scheduled tasks	Added support for scheduling tasks using cron. For more information, see Scheduled tasks (p. 524) .	7 June 2017
Spot Instances in the Amazon ECS console	Added support for creating Spot Fleet container instances within the Amazon ECS console. For more information, see Launching an Amazon ECS Linux container instance (p. 364) .	6 June 2017
Amazon ECS CLI v0.5.0	<p>New version of the Amazon ECS CLI released, which added the following features:</p> <ul style="list-style-type: none"> • Ability to push, pull, and list Amazon ECR images • Support for existing load balancers and Application Load Balancers in CreateService <p>For more information, see ECS CLI changelog.</p>	3 April 2017
Amazon SNS notification for new Amazon ECS-optimized AMI releases	Added ability to subscribe to SNS notifications about new Amazon ECS-optimized AMI releases.	23 March 2017
Microservices and batch jobs	Added documentation for two common use cases for Amazon ECS: microservices and batch jobs. For more information, see Common use cases in Amazon ECS (p. 718) .	February 2017
Container instance draining	Added support for container instance draining, which provides a method for removing container instances from a cluster. For more information, see Container instance draining (p. 428) .	24 January 2017
Docker 1.12 support	Added support for Docker 1.12. For more information, see Amazon ECS-optimized AMI (p. 333) .	24 January 2017
New task placement strategies	Added support for task placement strategies: attribute-based placement, bin pack, Availability Zone spread, and one per host. For more information, see Amazon ECS task placement strategies (p. 515) .	29 December 2016
Windows container support in beta	Added support for Windows Server 2016 containers (beta). For more information, see Amazon EC2 Windows containers (p. 842) .	20 December 2016
Blox OSS support	Added support for Blox OSS, which allows for custom task schedulers. For more information, see Scheduling Amazon ECS tasks (p. 509) .	1 December 2016
Amazon ECS event stream for CloudWatch Events	Amazon ECS now sends container instance and task state changes to CloudWatch Events. For more information, see Amazon ECS events and EventBridge (p. 632) .	21 November 2016

Change	Description	Date
Amazon ECS container logging to CloudWatch Logs	Added support for the awslogs driver to send container log streams to CloudWatch Logs. For more information, see Using the awslogs log driver (p. 283) .	12 September 2016
Amazon ECS services with Elastic Load Balancing support for dynamic ports	Added support for a load balancer to support multiple instance:port combinations per listener, which increases flexibility for containers. Now you can let Docker dynamically define the container's host port and the ECS scheduler registers the instance:port with the load balancer. For more information, see Service load balancing (p. 574) .	11 August 2016
IAM roles for Amazon ECS tasks	Added support for associating IAM roles with a task. This provides finer-grained permissions to containers as opposed to a single role for an entire container instance. For more information, see IAM Roles for Tasks (p. 699) .	13 July 2016
Amazon ECS CLI support for Docker Compose v2 format	The Amazon ECS CLI added support for Docker Compose v2 format. For more information, see ecs-cli compose (p. 102) .	8 July 2016
Docker 1.11 support	Added support for Docker 1.11. For more information, see Amazon ECS-optimized AMI (p. 333) .	31 May 2016
Task automatic scaling	Amazon ECS added support for automatically scaling your tasks run by a service. For more information, see Service auto scaling (p. 591) .	18 May 2016
Task definition filtering on task family	Added support for filtering a list of task definition based on the task definition family. For more information, see ListTaskDefinitions .	17 May 2016
Docker container and Amazon ECS agent logging	Amazon ECS added ability to send ECS agent and Docker container logs from container instances to CloudWatch Logs to simplify troubleshooting issues.	5 May 2016
Amazon ECS CLI v0.3 released	New version of the Amazon ECS CLI released, which added support for service creation with a load balancer.	11 April 2016
ECS-optimized AMI now supports Amazon Linux 2016.03.	The ECS-optimized AMI added support for Amazon Linux 2016.03. For more information, see Amazon ECS-optimized AMI (p. 333) .	5 April 2016
Docker 1.9 support	Added support for Docker 1.9. For more information, see Amazon ECS-optimized AMI (p. 333) .	22 December 2015
CloudWatch metrics for cluster CPU and memory reservation	Amazon ECS added custom CloudWatch metrics for CPU and memory reservation.	22 December 2015

Change	Description	Date
Amazon ECR	Added the new Amazon ECR service to the console, which added support for storing images that are controlled by resource-level permissions associated with Docker Hub or IAM users. Available in all AWS Regions, images are automatically replicated and cached globally so that starting hundreds of containers is as fast as a single container.	21 December 2015
New Amazon ECS first-run experience	The Amazon ECS console first-run experience added zero-click role creation.	23 November 2015
Task placement across Availability Zones	The Amazon ECS service scheduler added support for task placement across Availability Zones.	8 October 2015
Amazon ECS CLI with support for Docker Compose	The Amazon ECS CLI added support for Docker Compose.	8 October 2015
CloudWatch metrics for Amazon ECS clusters and services	Amazon ECS added custom CloudWatch metrics for CPU and memory utilization for each container instance, service, and task definition family in a cluster. These new metrics can be used to scale container instances in a cluster using Auto Scaling groups or to create custom CloudWatch alarms.	17 August 2015
UDP port support	Added support for UDP ports in task definitions.	7 July 2015
Environment variable overrides	Added support for deregisterTaskDefinition and environment variable overrides for runTask.	18 June 2015
Automated Amazon ECS agent updates	Added ability to see the ECS agent version that is running on a container instance. Also able to update the ECS agent from the AWS Management Console, AWS CLI, and SDK.	11 June 2015
Amazon ECS service scheduler and Elastic Load Balancing integration	Added ability to define a service and associate that service with an Elastic Load Balancing load balancer.	9 April 2015
Amazon ECS GA	Amazon ECS general availability in the US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), and Europe (Ireland) Regions.	9 April 2015

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.